НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Факультет Інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:

Завідувач кафедри

_____ Сергій СТІРЕНКО

“_____” _____ 2020р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп’ютерні системи та мережі»
спеціальності 123 «Комп’ютерна інженерія»
на тему: «Клієнт-серверна система для продажу туристичних турів»

Виконав: студент 4 курсу, групи ІО-63
Мельник Іван Миколайович _____

Керівник:
ст. викладач Алещенко Олексій Вадимович _____

Консультант з нормо-контролю:
проф. д.т.н. Сімоненко Валерій Павлович _____

Рецензент:
Радченко Костянтин Олександрович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ - 2020 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет Інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 123 «Комп'ютерна інженерія»
Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Сергій СТІРЕНКО

"____" _____ 2020р.

ЗАВДАННЯ
на дипломний проєкт студенту
Мельника Івана Миколайовича

1. Тема проєкту «Клієнт-серверна система для продажу туристичних турів», керівник проєкту Алещенко Олексій Вадимович, ст. викладач., затверджені наказом по університету від «7» травня 2020 р. № 1081-с.
2. Термін здачі студентом закінченого проєкту 16 червня 2020р.
3. Вихідні дані до проєкту: технічна документація, теоретичні дані, прототип робочого додатку.
4. Зміст пояснювальної записки: опис предметної області, дослідження аналогів, дослідження технології створення системи, інструкція користувача.
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо): структурна схема, ER-модель, функціональна схема
6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормо-контроль	Сімоненко В. П. проф.		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Затвердження теми роботи	01.09.2019	
2.	Вивчення та аналіз завдання	15.09.2019	
3.	Розробка архітектури та загальної структури cистем	16.10.2019	
4.	Розробка структур окремих підсистем	20.10.2019	
5.	Програмна реалізація системи	01.11.2019	
6.	Оформлення пояснювальної записки	01.05.2020	
7.	Передзахист	26.05.2020	
8.	Захист програмного продукту	16.06.2020	

Студент Іван МЕЛЬНИК

Керівник Олексій АЛЕЩЕНКО

Анотація

У дипломному проєкті були розроблені та реалізовані клієнт, сервер та база даних для системи з продажу туристичних турів. Система реалізує зручний інтерфейс, що дозволяє швидко та зручно здійснювати покупки. Також є можливість створювати власні поїздки, додавати до них додаткові послуги та акції. Для кращої взаємодії користувачів між собою та адміністрацією сайтів реалізована система online-чатів.

Для написання серверної частини була використана мова Java 8.0 та фреймворк Spring Boot. Клієнт був написаний використовуючи Angular 9.0. В якості системи управління базами даних був вибраний PostgreSQL. Розгорнута та протестована система була на хмарній платформі Heroku.

Annotation

In the thesis were developed and implemented a client server and database for the system for the sale of tourist tours. The system implements a user-friendly interface that allows you to quickly and easily make purchases. It is also possible to create your own trips, add additional services and promotions. An online chat system has been implemented for better interaction between users and the site administration.

Java 8.0 language and Spring Boot framework were used to write the server part. The client was written using Angular 9.0. PostgreSQL was chosen as the database. It was all deployed and tested on the Heroku cloud platform.

ОПИС АЛЬБОМУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проект	2	
2	А4	ІАЛЦ.467200.001 ВП	Відомість проєкту		
3	А4	ІАЛЦ.467200.002 ТЗ	Технічне завдання	4	
4	А4	ІАЛЦ.467200.003 ПЗ	Клієнт-серверна система для продажу туристичних турів	59	
5	А4	ІАЛЦ.467200.004 Д1	Структурна схема	1	
6	А4	ІАЛЦ.467200.005 Д2	ER-модель	1	
7	А4	ІАЛЦ.467200.006 Д3	Алгоритм обходу графа в ширину	1	
8	А4	ІАЛЦ.467200.007 Д4	Лістинг програми	10	
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					
21					
22					
23					
24					

Змн.	Арк.	ПІБ	Підпис	Дата	ІАЛЦ.467200.001 ВП		
Розробив		Мельник І.М..			Клієнт-серверна система для продажу туристичних турів. Відомість проекту	Літ.	Арк. 1 / Аркушів 1
Перевірив		Алещенко О.В.					
Н/Контр.		Сімоненко В.П.				КПІ ім. Ігоря Сікорського ФІОТ ІО-63	
Зав.каф.		Стіренко С.Г.					

ТЕХНІЧНЕ ЗАВДАННЯ

Зміст

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється для використання у курсі «Інженерія програмного забезпечення».

Область застосування: приклад при розробці схожих систем під час вивчення курсу «Інженерія програмного забезпечення».

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання кваліфікаційно-освітнього рівня «бакалавр комп'ютерної інженерії», затверджене кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського»

3. МЕТА І ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проекту є розробка клієнт-серверної системи для продажу туристичних турів

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література по комп`ютерним системам, публікації в періодичних виданнях, довідники на публікації в Інтернеті щодо даної теми.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розроблюваного продукту

- Самостійність курсу – чіткі контури предмету вивчення;
- Самодостатність – курс повинен містити необхідну інформацію та дані, які дозволяють в повній мірі розкрити мету курсу;
- Коректність та актуальність інформації, яку охоплює даний курс.

5.2. Вимоги до програмного забезпечення

- Операційна Unix-подібна або Windows 7/8/10;
- Java VM 8.0;
- PostgreSQL.

5.3. Вимоги до апаратно частити

- Комп'ютер на базі процесора Intel Core i3 та вище;
- Оперативної пам'яті не менше 4 Гбайт;
- Вільний простір жорсткого диску не менше 1 Тбайт.

6. ЕТАПИ РОЗРОБКИ

	Дата
Вивчення та аналіз завдання	15.09.2019
Розробка архітектури та загальної структури систем	16.09.2019
Розробка структур окремих підсистем	20.10.2019
Проектування програмного забезпечення	26.10.2019
Програмна реалізація продукту	01.11.2020
Тестування програмного забезпечення	04.04.2020
Налагодження та виправлення помилок	11.04.2020
Оформлення документації дипломної роботи	01.05.2020

Пояснювальна записка
до дипломного проєкту
на тему: «Клієнт-серверна система для продажу туристичних турів»

Київ – 2020 р

ЗМІСТ

Список скорочень

JVM – Java Virtual Machine. Віртуальна машина на якій запускаються програми написані на мові програмування Java.

PostgreSQL – об'єктно реляційна СУБД

Spring – програмний фреймворк для платформи Java

AOP – Aspect-oriented Programming, аспектно-орієнтоване програмування

Maven - фреймворк для автоматизації збирання проектів

Bundle - об'єднання товарів чи послуги разом, з метою продати їх як єдину комбіновану одиницю

Lifecycle – життєвий цикл

Angular - програмний фреймворк для платформи JavaScript

TypeScript – мова програмування що розширює можливості JavScript

HTML - HyperText Markup Language — мова розмітки гіпертексту

Apache Tomcat – контейнер сервлетів

Heroku – хмарна платформа

PaaS - Platform as a service, Платформа як сервіс

PWA - Progressive Web App, прогресивний веб-додаток

SPA – Single Page Application,

SSR – Server Side Rendering, рендер сторінок на стороні сервера

UX – User Experience, досвід користування

ВСТУП

Розвиток веб додатків набрав досить високого темпу і продовжую зростати. Технології, які були популярними кілька років тому стають застарілими уже зараз на зміну їм приходять ще більш досконалі. Розробники отримують можливість реалізувати нові функції якими приваблюють клієнтів тому попит на веб-додатки зростає з кожним днем. З поширенням автоматизації процесів все більше задач ми можемо робити в Інтернеті, тому для нас стала звичною можливість придбання авіаквитків чи бронювання номеру в готелі онлайн.

Існує досить багато схожих платформ, які можуть допомогти нам у вирішенні таких питань, але така різноманітність має свої недоліки. Сучасний користувач звик до швидкої роботи веб-додатків що вимагає правильно вибору стеку технологій, а через різноманітність наявних технології вибір підходящої для реалізації певної задачі може бути не таким простим. Недостатньо продуманий підхід під час проектування архітектури може вплинути не тільки на швидкість розробки а і на характеристики кінцевого продукту. Саме тому велика кількість платформ має низьку якість та не задовольняє потреби сучасних користувачів.

Попри те що веб додатки стали невід'ємною частиною нашого життя все ще дуже часто можна знайти їх досить незручними через необхідність використання декількох додатків для виконання різних задач спрямованих на одну ціль. Тому ще більш перспективним є ніша веб-додатків які поєднують в собі декілька інших що дозволяє ще легше та зручніше виконувати певні задачі.

Тому метою цієї роботи стала розробка системи яка не тільки дасть можливість купити авіаквитки і необхідних для подорожі сервіси які можуть запропонувати постачальники. Для виконання цієї задачі необхідно:

1. проаналізувати уже існуючі сервіси,

2. з’ясувати необхідні вимоги для реалізації,
3. проаналізувати існуючі фреймворки та вибрати належний,
4. розробити структуру для клієнт-серверної системи,
5. реалізувати клієнт-серверну систему.

РОЗДІЛ 1.
АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1. Постановка задачі

В теперішній час інформаційні технології є життєво необхідною для розвитку бізнесу галуззю. Інтернет став одним з основних джерел пошуку інформації тому для успішного бізнесу хороший веб-додаток є обов'язковою умовою. Веб-додаток будь-якої організації став синонімом до її обличчя тому йому приділяють досить багато уваги. Привабливий та дружелюбний інтерфейс допомагає в популяризації сервісу, а незручний та нагромаджений навпаки відлякує потенційних клієнтів. Тому питання в створенні хорошого веб-додатка є достатньо складним, а висока конкуренція тільки додає необхідності в створенні чогось унікального, що легко може привабити покупців.

З такої точки зору привабливою сферою є створення додатків, які надають можливість клієнту купувати туристичні поїздки і одночасно надають можливість різним постачальникам презентувати свої послуги.

Для того щоб створити таку систему варто визначити критерії які ляжуть в її основу. Завдяки постійному користуванню Інтернетом ми можемо скласти мінімальний список параметрів які є ключовими для веб-додатку такого типу базуючись лише на власному досвіді. В такий список увійдуть такі пункти як:

- Пошук та покупка туристичного туру та додаткових сервісів для нього;
- Можливість презентувати свої поїздки та додаткові сервіси для них постачальникам;
- Зворотний зв'язок з користувачем через online-чат та систему коментарів;
- Швидке сповіщення користувачів про ключові події;
- Збір статистики про популярність та напрямки використання платформи для адміністраторів системи;

- Зручний та інтуїтивно зрозумілий інтерфейс;
- Адаптивну структуру для роботи на мобільних платформах;
- Працездатність на найбільш популярних платформах.

Проте створювати систему базуючись лише на власному досвіді скоріш всього не призведе до успіху тому далі слід проаналізувати уже існуючі системи та змінити початковий список додавши до нього нові та відсіявши зайві вимоги.

1.2. Аналіз існуючих аналогів

В нинішньому світі існує безліч сайтів основною метою яких є зручне донесення інформації до користувача. Але правильно виконати поставлену задачу не завжди так просто. Для цього при створенні сервісу необхідно проаналізувати сучасні підходи до розробки, правильно задати структуру проекту, вибрати цільову аудиторію користувачів та спрогнозувати навантаження на майбутню систему. Тому для проведення аналізу веб-додатків будемо використовувати такі критерії:

- User Experience (UX);
- Сучасний дизайн представлений в одному стилі;
- Якість реалізації мінімального набору необхідних функцій;
- Наявність додаткових чи унікальних функцій;
- Використання новітніх технологій під час розробки;
- Технічні характеристики.

User Experience – це досвід який користувач отримує під час використання будь-якого продукту, наприклад веб-сервісу. Хорошим UX можна назвати коли після користування системою ми можемо дати позитивні відповіді на такі питання: Чи хочеться використовувати цей сервіс знову? Чи був корисним цей сервіс? Чи виконує він поставлені йому задачі?

Дуже важливо розуміти що хороший UX це один з найкращих способів отримати довіру та лояльність свої користувачів. Не слід забувати і про те що

стандарти UX зростають з кожним днем і користувачі звикають до кращих умов та стають менш толерантними до поганих.

Про те що необхідно підтримувати нові тенденції в стилістиці навряд необхідно нагадувати, проте все ще існує багато сервісів які не оновлювали свій дизайн на протязі багатьох років. Велика кількість користувачів в наш час може просто нехтувати такими сервісами і навіть не починати користуватись ними так як вони упереджено думають що вони застарілі та незручні.

Вибір сучасних технологій при розробці програмного додатку є не менш важливим етапом. Реалізація нових тенденцій в сфері UX вимагає сучасних архітектурних рішень, що, в свою чергу, спонукає до використання нових технологій. Не залежно від розмірів додатку на сьогоднішній день можна знайти найбільш підходящі програмні засоби та рішення що допоможуть максимально ефективно розробляти, вдосконалювати та оптимізувати систему. Але не слід підходити до вибору легковажно. При виборі фреймворку на якому буде побудована система слід пам’ятати про такі характеристики як:

- Розвиток та наявність довготривалої підтримки версій;
- Велика та активна аудиторія уже існуючих користувачів та якісна документація;
- Підтримка сучасних архітектурних рішень «з коробки».

Розглянемо існуючі аналоги запропонованої системи такі як TopTour, ANEX Tour, Join UP, G Advetures.

1.2.1. TopTour

На рис. 1.1 зображено початкову сторінку веб-додатку туристичного агентства TopTour де користувачу пропонують відразу перейти до основної задачі – пошуку бажаної поїздки.

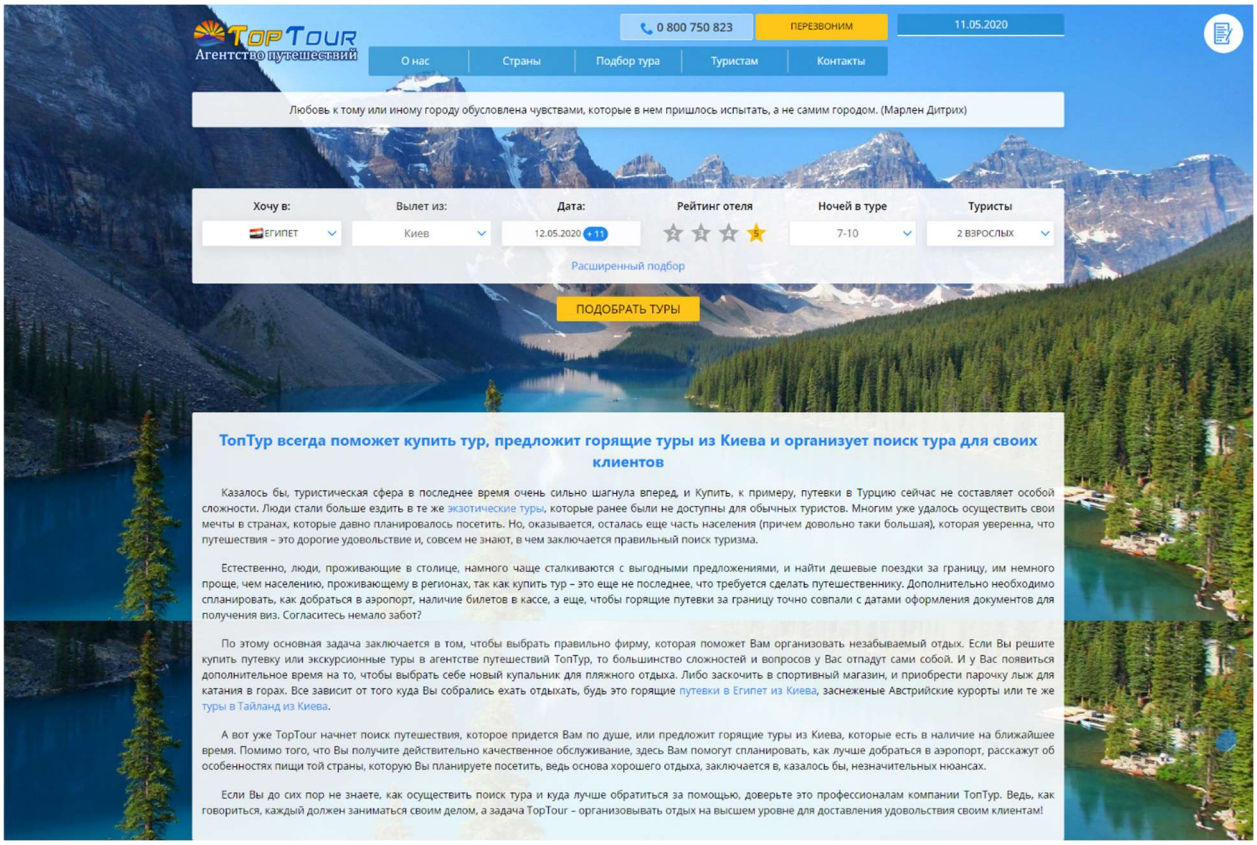


Рис. 1.1. Головна сторінка веб-сервісу «TopTour»

Для цього необхідно вибрати такі параметри:

- Пункт прибуття туристичного туру;
- Пункт відправлення туристичного туру;
- Дата коли відбудеться сам туристичний тур;
- Кількість днів;
- Кількість людей.

Також присутній більш детальний пошук який дозволяє здійснити детальній підбір поїздки по окремим регіонам країни та обмежити підбірку по ціні. Незважаючи на те що дизайн веб-додатку є доволі простим він не вписується в сучасні рамки. В ньому присутні елементи що відрізняються стилями що відразу помічається, це добре видно на рис. 1.2.

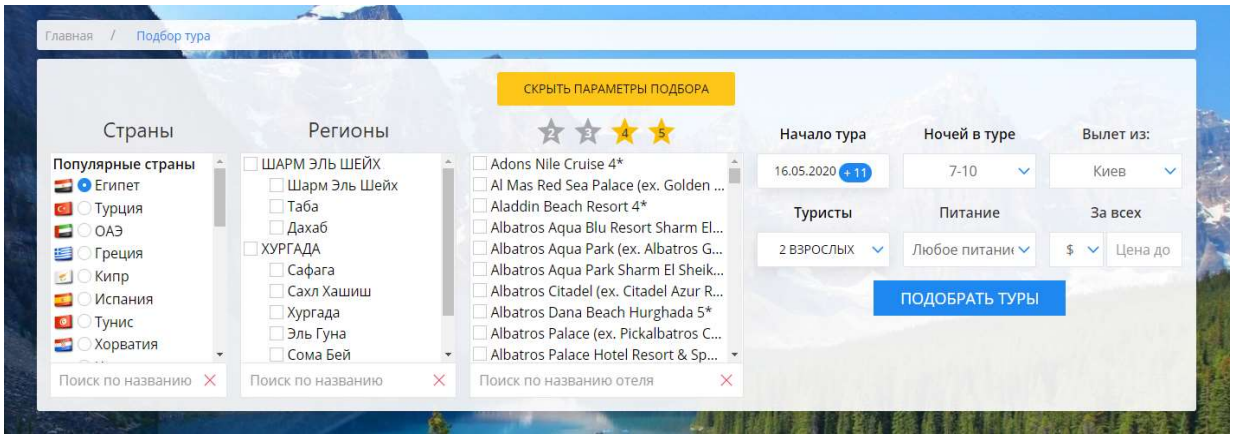


Рис. 1.2. Пошуковий фільтр

Відсутня система відгуків інших користувачів що не дає змоги повноцінно оцінити якість як окремого тури так і всієї системи в цілому. З технічної точки зору веб-додаток відповідає сучасним нормам швидкості але це досягнуто за рахунок невеликої кількості сторінок та матеріалу розміщеного на них. Побудований додаток на архітектурі MPA що стає зустрічатись все рідше проте досить міцно тримає свої позиціє при побудові невеликих та мало функціональних веб-додатків.

1.2.2. G Adventures

Розглянемо G Adventures (рис. 1.3) - це відома компанія що займається туристичними турами по всій планеті чим і відрізняється від більшості конкурентів. Це стало можливо завдяки тому, що була реалізована функція не тільки покупця а і організатора поїздок. Такий підхід дозволяє не тільки збільшити пропозицію а і привернути більше клієнтів так як більше непотрібно використовувати декілька сервісів для пошуку.

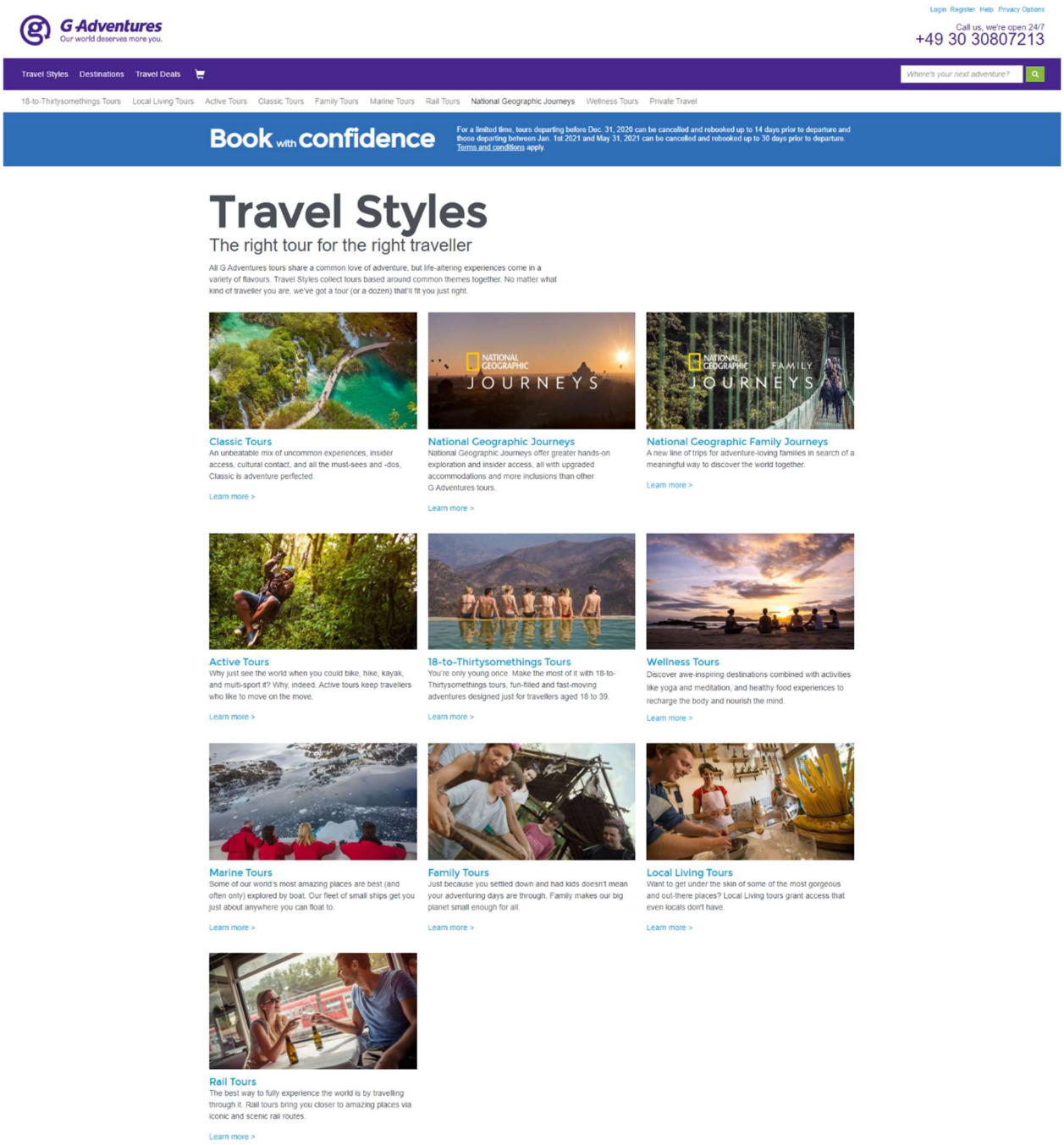


Рис. 1.3. Головна сторінка веб-сервісу «G Adventures»

Проте великий масштаб бізнесу та довгий час на ринку має і негативну сторону. Основа на якій був побудований додаток уже морально застаріла та починає не справлятись з навантаженням. Швидкість завантаження навіть сторінок з загальною інформацією є відносно повільною та може досягати 6с, що видно на рис. 1.4.

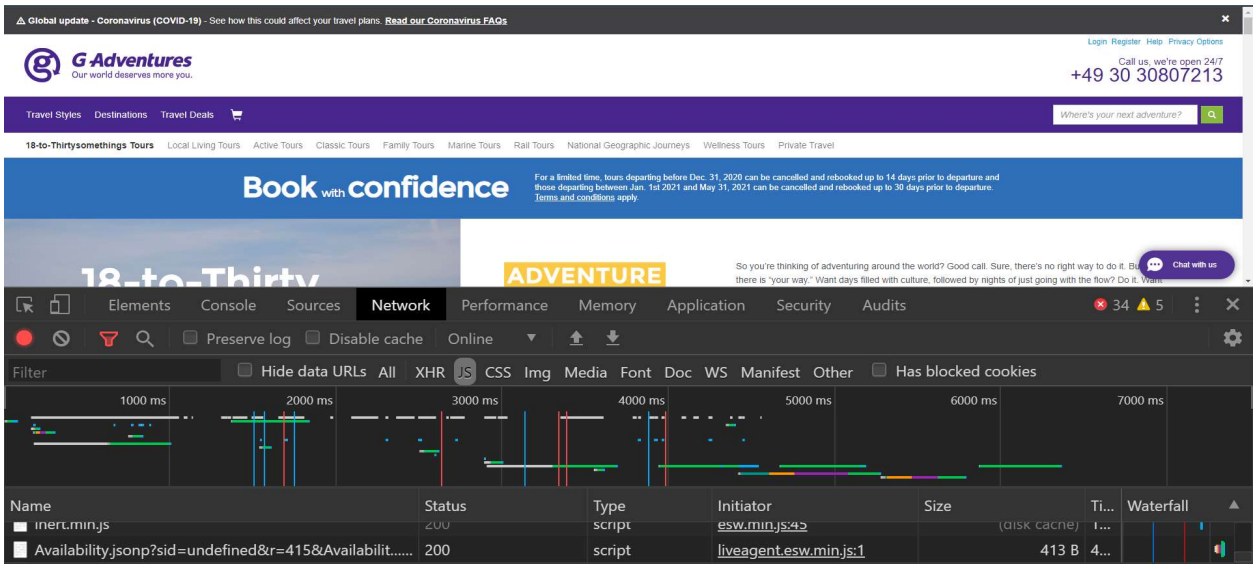


Рис. 1.4. Швидкість завантаження сторінки G Adventures

При проектуванні додатку як і в попередньому приклади використано MPA. Але в даному випадку це стало одним з вузьких місць що погіршують UX який отримують користувачі.

1.2.3. Aviasales

Aviasales (рис. 1.5) є прикладом добре оптимізованої системи де використання SPA архітектури принесло свої плоди.

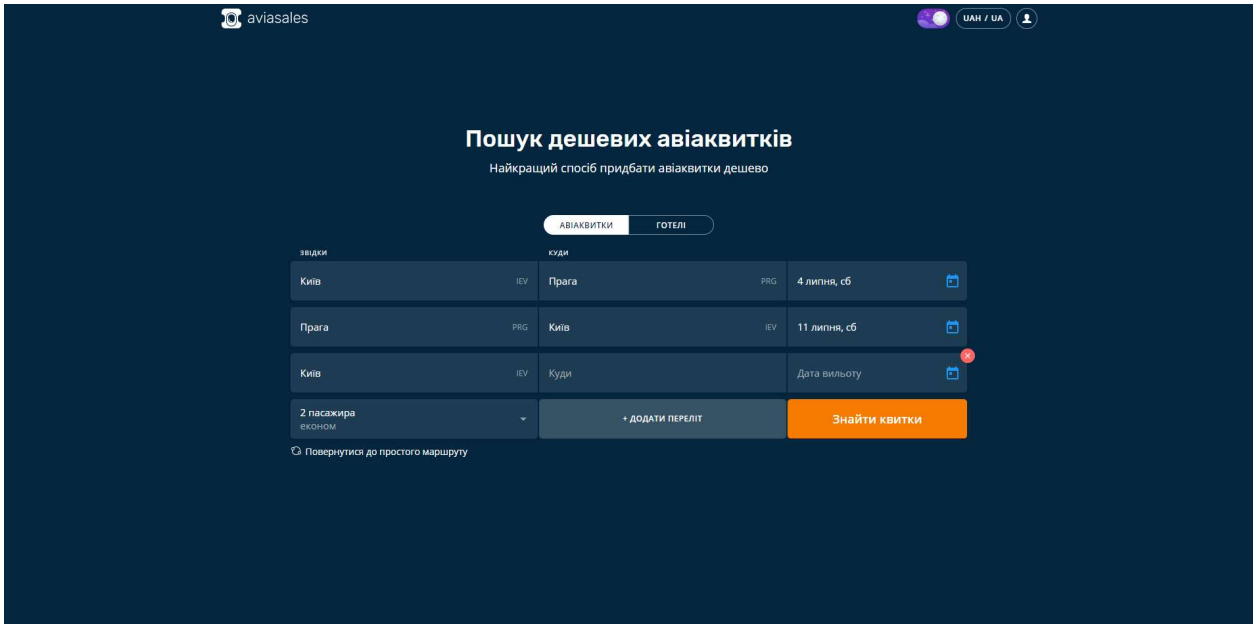


Рис. 1.5. Головна сторінка веб-сервісу Aviasales

Проте це не звичайний додаток де представлена можливість купити туристичний тур, це своєрідний пошуковик який використовує дані з багатьох інших сервісів.

Чудово реалізована адаптація до різних браузерів та платформ тому не виникне ніяких проблем при користуванні на будь-яких форматах пристроїв.

Ще одним достоїнством сервісу є дуже детальна інформація про вибрані поїздки. Можна дізнатись відразу про тривалість всіх відрізків подорожі, наявність Wi-Fi та USB-розетко під час подорожі, тощо.

Мінусом цього додатку є відсутність можливості додати будь-який сервіс до поїздки.

1.2.4. Join UP

Розглянемо ще одну систему для продажу туристичних поїздок – Join Up (рис. 1.6).

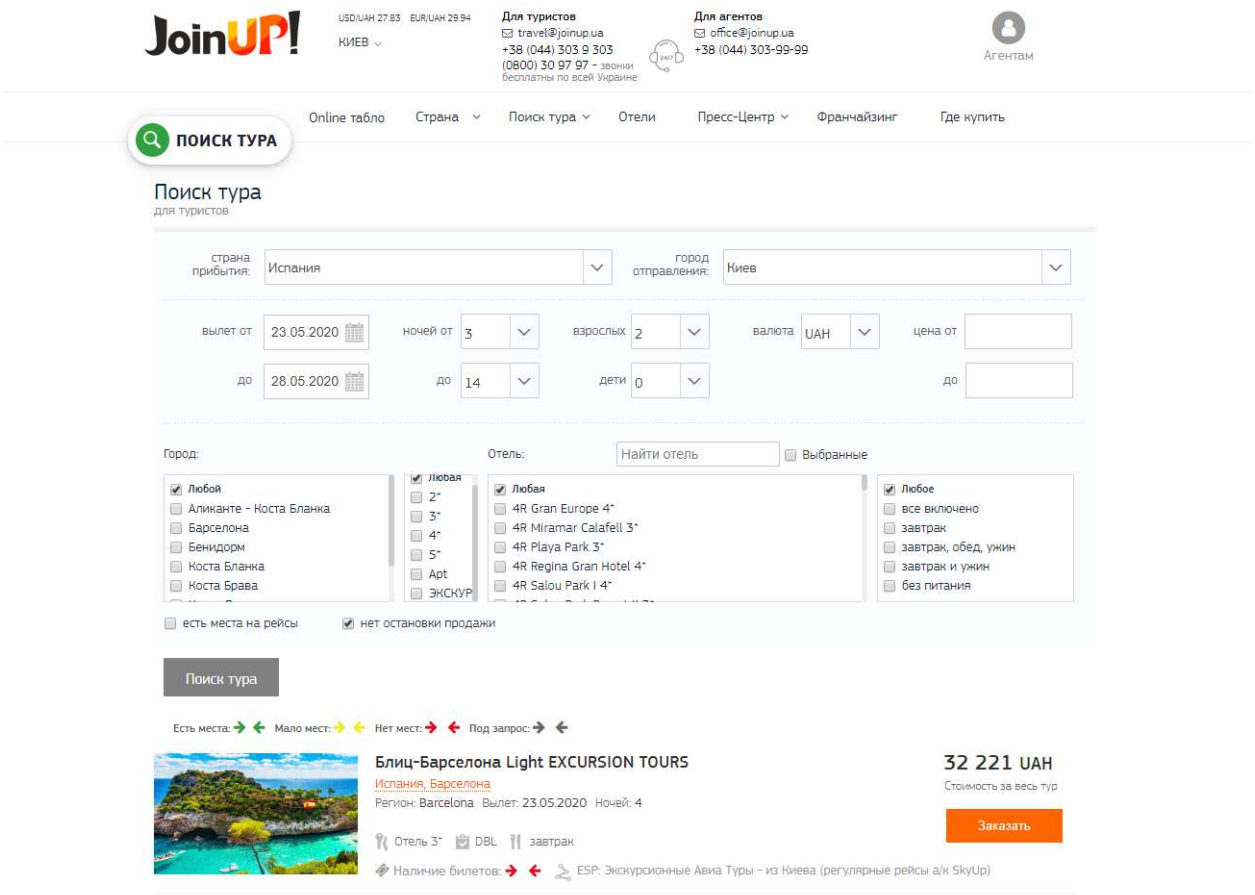


Рис. 1.6. Пошукова сторінка «Join Up»

Це велика компанія яка входить до франшизи що забезпечує велику клієнтську базу. Свою популярність сервіс також отримав за широкий вибір та добре побудований зв'язок з користувачами. Реалізована можливість отримати відповідь на питання що турбують юзера через дзвінок, e-mail лист та online-чат. Також є розсилка повідомлень про появу нових турів чи зміну в наявних.

В системі реалізована можливість створення «Агента» який зможе створювати поїздки. Окрім самої поїздки є додаткова опція у виді вибору готелю що є несумнівним плюсом для такої системи.

Проте мінусом системи є недостатня багатоплатформність а на моніторах з широкою діагоналлю екрану користуватись веб-додатком не зовсім зручно.

1.2. Формулювання вимог

На основі проведеного аналізу, виділивши пункти що є перевагами, можна сформулювати більш детальні вимоги до клієнт-серверної системи.

Створена система повинна давати змогу знайти туристичний тур за допомогою пошуку по зручним критеріям. Пошук повинен видавати не просто результати які відповідають пошуковим критеріям а і рекомендувати поїздки за кращою ціною чи найбільшою популярністю.

Необхідно реалізувати функціонал не тільки для користувача який матиме змогу купувати поїздки а і для перевізників. Вони будуть постачати нові туристичні поїздки та додаткові сервіси. Для підвищення якості подорожей які створюватимуть провайдери необхідно ввести спеціальний тип користувачів який буде перевіряти правильність та достовірність путівок. Також таким користувачам буде відведена роль в вирішенні проблем які можуть виникнути під час поїздки або після неї.

Для користувачів, в свою чергу, є необхідним створення зручного інтерфейсу в якому вони зможуть задати питання які їх цікавлять та отримати допомогу при необхідності.

Не слід забувати і про підтримку активності у користувачів що уже користувались сервісом. Тому вагомими є листи на пошту та швидкі сповіщення під час користування веб-додатком з новинами від провайдерів.

Окремо варто наголосити на створенні бандлів (bundles) які об’єднують в собі декілька послуг. Це дозволить зручно купувати декілька послуг які пов’язані між собою.

Для забезпечення можливості системи працювати довгий час без серйозних змін в коді необхідно вибрати правильний підхід до побудови архітектури. Приймаючи до уваги те що буде реалізована можливість додавання нових подорожей іншими провайдерами та реалізацію додаткового функціоналу для користувача використовувати MPA буде недоцільно.

ВИСНОВОК ДО РОЗДІЛУ 1

В цьому розділі була обґрунтована актуальність вибраного напрямку та сформульована мета дипломного проекту. Проаналізувавши сучасний ринок були сформульовані початкові вимоги до реалізації майбутньої системи. Для того щоб підвищити конкурентоздатність системи необхідно проаналізувати існуючі аналоги на предмет виявлення недоліків та переваг що і було зроблено. Взявши в приклад популярні сервіси зі схожою тематикою були відібрані нові та деталізовані початкові вимоги по функціональному наповненню клієнт-серверної системи.

РОЗДІЛ 2.
АНАЛІЗ ІНСТРУМЕНТІВ ДЛЯ СТВОРЕННЯ КЛІЄНТ-СЕРВЕРНОЇ СИСТЕМИ

2.1. Java

Мова програмування грає важливу роль при створенні будь-якого проекту тому до її вибору необхідно поставитись досить уважно. В даній роботі була використана дуже відома та популярна мова Java. Java - це сучасна мова програмування, розроблена компанією Sun Microsystems (зараз її купила Oracle). Особливо його не слід плутати з JavaScript (мова скриптів, що використовується в основному на веб-сайтах), оскільки Java не має нічого спільного. Однією з його найбільших сильних сторін є його відмінна портативність: після створення вашої програми вона автоматично працюватиме в Windows, Mac, Linux та ін. Основними характеристиками цієї мови є те що вона паралельна, строго-типізована, об’єктно орієнтована на основі класів. Зазвичай вона компілюється в байт-код та бінарний формат визначений віртуальною машиною JVM [5].

Java простіша, з набагато більш читаним синтаксисом, ніж C, C ++ або будь-яка інша мова. Java ідеально підходить для вивчення об'єктно-орієнтованого програмування, але набагато менше для процедурного програмування, де C набагато краще. Об'єктно-орієнтоване програмування - корисна навичка, оскільки допомагає керувати складністю програми в реальному світі. Простіше мислити з точки зору класу та предметів.

У Java є багатий API, і у вас є більше можливостей, включаючи графіку та звук. Ви, очевидно, все це можете зробити. з іншими мовами, але вас часто змушуватимуть завантажувати та встановлювати різні модулі та бібліотеки, що для початківців є складним завданням. Під час встановлення Java більшість цих функцій включаються автоматично. Java підтримується дуже великою спільнотою, і незалежно від того, який тип питань, сумнівів чи проблем у вас є Інтернет може допомогти вам знайти відповіді.

2.2. Spring

Spring є базою для розробки додатків, в основному для компаній, але не обов'язково. Він надає безліч функцій, які іноді є зайвими або які можна налаштувати або використовувати декількома способами: це залишає розробника вільним у виборі рішення, яке йому найбільше підходить та яке відповідає потребам. Таким чином, Spring є одним з найпоширеніших фреймворків у світі Java: його популярність зросла на користь складності Java EE, особливо до версії 5, але також завдяки якості та багатству функцій, які він пропонує. Ядро, що опирається на контейнер типу IoC, забезпечує управління життєвим циклом бінів (beans) та ін'єкцію залежностей. Також фреймворк полегшує інтеграцію з багатьма проектами з відкритим кодом або API. Spring фреймворк був спочатку звичайною прикладною програмою, але тепер це справжня платформа що охоплює багато потреб. Spring дає велику гнучкість у функціях та проектах, які використовуються в додатку. Наприклад, можна використовувати контейнер Spring для управління бінами в основному без використання AOP. Зараз Spring асоціюється з поняттям простого та легкого способу побудови проекту на відміну від складних додатків написаних на основі Java EE.

2.2.1 Spring Boot

Spring є основою, добре відомою розробникам Java за багатьма функціями, які вона надає в Інтернеті, безпеці і навіть доступі до даних. Але він, на жаль, відомий своєю конфігурацією, яка може бути складною і виснажливою. Не рідкістю було витратити кілька днів на конфігурацію проекту Spring, особливо для людей, які не знайомі з використанням фреймворку. Виходячи з цього спостереження, команди Spring вирішили працювати над проектом для сприяння розробці Spring-програм для розробників. Так народився Spring Boot [7].

Spring Boot - це фреймворк який має на меті полегшити конфігурацію проекту Spring і скоротити час, відведений на початок проекту.

Для досягнення цієї мети Spring Boot базується на кількох елементах:

- Веб-сайт, який дозволяє швидко генерувати структуру вашого проекту, включивши всі залежності Maven, необхідні для вашої заявки. Цей спосіб також доступний через плагін Eclipse STS;
- Використання лаунчерів для управління залежностями. Spring згрупував Maven залежності Spring's в "мегазалежності", щоб полегшити управління ними. Наприклад, якщо ви хочете додати всі залежності для управління безпекою, просто додайте стартер "spring-boot-starter-security";
- Автоконфігурація, яка застосовує конфігурацію за замовчуванням під час запуску програми для будь-яких залежностей, наявних у ній. Вона активується в момент, коли ви використовуєте в своїй програмі анотацію "@EnableAutoConfiguration" або "@SpringBootApplication".

Звичайно, ви можете перевантажена за допомогою попередньо визначених властивостей Spring або через конфігурацію Java. Автоконфігурація спрощує налаштування, не обмежуючи функціональність Spring. Наприклад, якщо ви використовуєте стартер "spring-boot-starter-security", Spring Boot налаштує безпеку у вашій програмі, включаючи користувача за замовчуванням та пароль, що генерується випадковим чином при запуску програми.

На додаток до перших елементів, що полегшують конфігурацію проекту, Spring Boot пропонує й інші переваги, зокрема щодо розгортання програми. Зазвичай для розгортання програми Spring потрібна генерація .war-файлу, який повинен бути розгорнутий на сервері, як Apache Tomcat. Spring Boot спрощує цей механізм, пропонуючи можливість безпосередньо інтегрувати сервер Tomcat у свій виконуваний файл. Після запуску цього буде запущений вбудований Tomcat для запуску вашої програми [13].

Нарешті, Spring Boot надає можливість отримати метрики, які вони можуть дотримуватися, коли додаток розгорнуто у виробництві. Для цього

весняний завантажувач використовує "Actuator", який є системою, що дозволяє контролювати додаток за допомогою певних URL-адрес або команд, доступних через SSH. Таким чином визначити власні показники можна дуже легко. Ось неповний список показників, доступних за замовчуванням:

- метрики: показники програми (процесор, пам'ять, ...);
- біни: список бінів які використовує Spring;
- trace: список HTTP-запитів, надісланих до програми;
- tread dump: список поточних потоків які наявні в запущенів програмі;
- health-check: деякі параметри по яким можна визначити працездатність компонентів системи;
- env: список профілів, властивостей та змінних середовищ.

2.3. JavaScript and TypeScript

Javascript - одна з найбільш використовуваних комп'ютерних мов для розробки веб-сторінок, дуже гнучка і має чудову функціональність. TypeScript це в основі javascript в який додали набір розширень. Розглянемо переваги та недоліки двох мов програмування.

JavaScript - мова програмування, що інтерпретується динамічним та розширеним рівнями. Це одна з трьох основних технологій всесвітньої павутини з HTML та CSS. Більшість веб-сайтів використовують його, і це підтримується всіма сучасними веб-браузерами. JavaScript - прототип, заснований на функціях першого класу, що робить його універсальним інструментом, який підтримує функціональні та об’єктно-орієнтовані стилі програмування. Він включає API для роботи з таблицями, виразами та текстом, але не має підтримки вводу чи виводу, наприклад, сховища, медіа, мережі та графічних установок. Для цього вам потрібно покластися на середовище хоста, але є такі фреймворки, як Node.js, які дозволяють скористатися всіма перевагами JavaScript в повній мірі. JavaScript також застосовується в додатках, які не повністю базуються на веб-сторінках, таких як віджети на робочому столі та веб-переглядачі. Він також відіграє важливу

роль у розробці ігор, офісних та мобільних додатків та мережевого програмування на стороні сервера. Розвиток JavaScript ще більше покращився з появою нових різноманітних фреймворків. Для розробки веб-сторінок на сервері ви можете використовувати безліч мов та фреймворків, але для програмування на стороні клієнта. JavaScript - єдиний варіант, оскільки він підтримується всіма браузерами. Для подолання цієї проблеми були розроблені новіші мови які можна скомпілювати в JavaScript. Ці нові мови пропонують нові поняття та покращений синтаксис, приховуючи складні моменти JavaScript. Вони можуть запропонувати хороші способи написання та керування великим кодом, при цьому автоматично генеруючи більш правильний JavaScript як альтернативу [8].

Typescript - це новий конкурент, розроблений корпорацією Майкрософт та розроблений для підтримки першокласної підтримки громадян у Visual Studio, включаючи інтелект та помилки компіляції під час польоту. TypeScript відрізняється від Coffeescript тим, що це повний набір JavaScript. Весь код JavaScript є повністю дійсним кодом TypeScript. TypeScript надає велику допомогу в створенні кращого JavaScript, а також природний синтаксис для визначення класів. Основна перевага TypeScript - це додаткова підтримка для перевірки статичного типу, що полегшує взаємодію із зовнішніми бібліотеками JavaScript, або без перевірки типу, або із створеним вами визначенням інтерфейсу. TypeScript добре інтегрується з усіма існуючими кодами JavaScript і його легко зрозуміти, навіть якщо ви знаєте трохи JavaScript та C. Ще один момент, який слід вибрати TypeScript, - це те, що одного дня TypeScript сьогодні буде стандартний JavaScript і тому буде виконуватися в браузері. TypeScript дозволяє створювати сьогоднішні веб-програми з мовними особливостями завтра. TypeScript - єдиний, який є дуже суворим набором JavaScript. Це означає, що будь-який тип дійсного коду JavaScript також хороший для TypeScript і, безумовно, сумісний з існуючими бібліотеками.

Переваги TypeScript:

1) Наявність JavaScript. Кожна існуюча програма виконана на мові Javascript вже є дійсною програмою TypeScript, яка надає їй найкращу підтримку для існуючих бібліотек, що особливо корисно, якщо вам потрібно інтегруватися з існуючою базою коду JavaScript;
2) Розроблений та підтримується Microsoft. Будучи створеним Microsoft, TypeScript набагато частіше, ніж більшість інших подібних проектів з відкритим кодом, отримують постійну довгострокову підтримку, хорошу документацію та постійний потік розвитку;
3) Має велику кількість адаптованих бібліотек включаючи jquery, angular, bootstrap, d3, lodash та ще багато іншого.

Недоліки TypeScript:

1) Занадто багато схожого на JavaScript. Він має деякі переваги перед JavaScript, але оскільки він розроблений на основі JavaScript, це означає, що всі погані частини JavaScript все ще присутні;
2) Не зовсім явна типізація. Типовим типом при оголошенні та використанні змінної є тип “any”;
3) Немає структури пакетів Java. Якщо ви віддаєте перевагу підходу Java для розподілу вашого коду на різні пакети, система модуля набору текстів вас трохи заплутає.
4) Немає підтримки умовної компіляції;
5) Немає підтримки для усунення невикористовуваного коду. Типографічний компілятор не видаляє код із створених файлів, для видалення невикористаного коду після компіляції потрібно використовувати зовнішні інструменти. Що складніше, оскільки компілятор уже усунув всю інформацію про тип.

2.4. HTML and CSS

HTML - це універсальна мова, що використовується на веб-сторінках, які читаються усіма веб-браузерами (Internet Explorer, Netscape, Mozilla тощо).

Ця мова працює відповідно до складання та комбінації тегів, що дозволяють структурувати та надавати бажаний зовнішній вигляд тексту, зображень та мультимедійних даних відповідно до потрібного макета. Її розвиток починається з далеких часів коли Інтернет був просто текстом, що було трохи нудно. На щастя, пройшло багато часу, перш ніж було додано можливість вставляти зображення (та інші типи цікавого вмісту) у веб-сторінку.

HTML не є мовою програмування: це мова розмітки, яка використовується для того, щоб розповісти браузеру, як структурувати відвідувані веб-сторінки. Це може бути настільки складно або просто, як веб-розробник хоче. HTML складається з серії елементів, за допомогою яких ви можете створювати, обертати або тегувати різні частини вмісту, щоб вони відображалися чи діяли певним чином. Навколишні теги можуть перетворити невеликий вміст у посилання на іншу сторінку в Інтернеті, розмістити слова курсивом тощо.

По суті, HTML - це дійсно проста мова, що складається з елементів, які можна застосувати до фрагментів тексту в документі, щоб надати їм інше значення (це абзац? Чи це список з позначеннями? Чи це частина таблиці?), щоб структурувати документ у логічні розділи та інтегрувати вміст, як зображення чи відео на сторінці.

Написаний код на даній мові програмування є досить простим для сприйняття. Проте інколи містить занадто багато лишнього тексту через громіздкі назви тегів, а можливість створення власних тегів лише погіршує ситуацію. На рисунку 2.1 приведено приклад коду написаний за допомогою розглянутої мови програмування.

CSS мова з каскадним стилем що використовує стилістичні елементи схожі до HTML-коду. Використовуючи CSS, можна змінити текстовий текст, зробити шрифт, шрифт та можливість ввести параметри. В початкових версіях HTML розробники сайтів стикалися з великими проблемами пов’язаними з налаштуванням стилів. Це збільшувало час витрачений на розробку що в свою

чергу збільшувало ціну. Таким чином, CSS був створений для вирішення цієї проблеми.

```
<div class="container">
  <div class="card card-layout">

    <mat-tab-group>
      <mat-tab *ngIf="authority == 'ROLE_USER'" label="Support chat">
        <app-support-chat></app-support-chat>
      </mat-tab>
      <mat-tab *ngIf="authority == 'ROLE_APPROVER'" label="Approvers chat">
        <app-approvers-chat></app-approvers-chat>
      </mat-tab>
      <mat-tab label="Global chat">
        <app-global-chat></app-global-chat>
      </mat-tab>
    </mat-tab-group>

  </div>
</div>
```

Рис. 2.1. Приклад html-сторінки

Зв'язок між HTML та CSS дуже міцний. Оскільки HTML є мовою розмітки (основа веб-сайту), а CSS орієнтований на стиль (всю естетику веб-сайту), вони йдуть разом. CSS технічно не є необхідністю, але ви, мабуть, не хотіли б шукати веб-сайт, який використовує лише HTML, оскільки він здасться повністю занедбаним.

Перш ніж використовувати CSS, усі стилі повинні були бути включені до розмітки HTML. Це означає, що слід окремо описувати весь фон, кольори шрифту, вирівнювання тощо. Але CSS дозволяє стилювати все в іншому файлі, тим самим створюючи стилі окремо. А згодом інтегруючи файл CSS у верхній частині розмітки HTML. Це робить розмітку HTML чистою та простою у обслуговуванні.

Переваги CSS:

- Різниця між веб-сайтом, який реалізує CSS, і тим, хто його не використовує, величезна і помітна;
- CSS дозволяє стилювати все в інший файл, тим самим створюючи стиль окремо. А згодом інтегруйте файл CSS у верхній частині розмітки HTML. Це забезпечує розмітку HTML чистою та простою у обслуговуванні;
- CSS дозволяє мати декілька стилів на HTML-сторінці, що робить можливості налаштування майже нескінченними. Сьогодні це стає більше необхідністю, ніж простим ресурсом.

CSS - це дуже потужний інструмент, який дозволяє створити кілька функціональних можливостей замість використання JavaScript або іншої, важчої, мови. Якщо використовувати їх правильно, CSS може забезпечити чудовий досвід для розробника та користувачів веб-сторінок.

За допомогою каскадних таблиць стилів можна створювати складні анімації, створювати ефекти за допомогою паралакса, завдяки чому здається, що фонове зображення має глибину, створювати інтерактивні веб-сайти, а також ігри з HTML5 та CSS3.

2.5. Angular

Angular - це фреймворк з відкритим кодом, розроблений Google для полегшення створення та програмування веб-додатків по принципу SPA.

Angular повністю розділяє клієнтський і серверний додаток у програмі, уникає написання повторюваного коду та підтримує все в належному порядку за допомогою шаблону MVC.

У SPA-архітектурі швидкість завантаження може бути дуже повільною під час першого відкриття, проте навігація після цього є абсолютно миттєвою, оскільки вже завантажена вся сторінка.

Серед інших переваг, фреймворк є модульним та масштабованим, адаптуючись до наших потреб і базуючись на стандарті веб-компонентів, а за

допомогою набору інтерфейсів дозволяє нам створювати нові персоналізовані теги HTML, які можна використовувати повторно.

Основною мовою програмування Angular є Typescript. Однією з важливих характеристик Angular є реактивне програмування, забезпечує автоматичне оновлення елементів відразу після внесення необхідних змін.

2.7. Heroku

Поява хмарних обчислень та збільшення кількості послуг, які пропонують хмарні платформи, значно полегшили процес розробки та розгортання веб-систем. В даний час розробник має в своєму розпорядженні кілька варіантів послуг у хмарі. Вони охоплюють кілька типів додатків, від простого веб-сервісу до складного додатка. Пропозиції також відрізняються залежно від рівня дозволеного налаштування, при цьому можливе цитування від повністю настроюваних інфраструктурних рішень до платформ, готових до прийому програми.

У цьому сценарії Heroku завойовує частку ринку як платформа як постачальник послуг (PaaS). Цей тип рішення абстрагує розробника від деталей інфраструктури, роблячи доступними контейнери для встановлення програм, полегшуючи технічне обслуговування, розширення та масштабованість, на додаток до пропонованої більшої спритності зробити додаток доступним в Інтернеті з меншими початковими витратами.

Завдяки цим атракціонам рішення PaaS привертали увагу головним чином до невеликих проектів через низьку початкову вартість, оскільки більшість постачальників пропонують безкоштовний план, що дозволяє розгорнути додатки, але з обмеженнями на використання таких ресурсів, як процесор, пам'ять, мережа тощо.

Виходячи з цього, Heroku як постачальник PaaS, безсумнівно є лідером.

Heroku належить до категорії хмарних обчислювальних служб, відомих як PaaS, в якій постачальник доставляє замовнику середовище, готове отримати заявку. На відміну від IaaS, в якій клієнт наймає машини (реальні або

віртуальні) і відповідає за встановлення бібліотек, збирання структур файлової системи, серед інших ресурсів, PaaS - це рішення високого рівня.

Оскільки середовище доставляється постачальником, замовник просто повинен зосередитися на розробці та встановленні програми.

В даний час існує безліч варіантів для постачальників послуг PaaS, крім самого Heroku. В якості прикладу можна згадати RedHat OpenShift, AWS Elastic Beanstalk, Jelastic, CloudBees. В основному те, чим відрізняється кожен, - це технології, які вони дозволяють використовувати як мови чи контейнери кількість додаткових послуг і, звичайно, ціноутворення, тобто вартість на пам'ять, процесор та диск кожного екземпляра програми. Ще одна змінна - простота створення та встановлення програми. На даний момент Heroku є на крок попереду інших постачальників, завдяки простоті створення, обслуговування та масштабування програми.

Heroku було створено в 2007 році трьома північноамериканськими розробниками: Джеймсом Лінденбаумом, Адам Віггінс та Оріоном Генрі. Спочатку підтримка була лише для додатків, розроблених у Ruby, що працюють на веб-сервері Rack. У 2010 році проект був придбаний компанією Salesforce, а в 2011 році він розпочав процес підтримки інших мов та рамок. В даний час Heroku підтримує Ruby, Java, Clojure, Python, Scala і Node і більш ніж три мільйони встановлених додатків.

Що стосується фізичних установок, Heroku використовує консолідовану інфраструктуру Amazon та працює в регіоні США (куди входять центри обробки даних у Сполучених Штатах), а в регіоні ЄС (центри даних, розташовані в Ірландії) знаходиться в бета-версії. Що стосується доступності, незалежно від регіону, звіти, що відображаються на самому веб-сайті, гарантують, що Uptime за останні 12 місяців завжди перевищував 99%.

На відміну від послуги IaaS, яка забезпечує клієнта цілою машиною, щоб він міг налаштувати послуги та встановити свою програму, Heroku, як і інші послуги PaaS, забезпечує середовище виконання програми. Цей тип рішення

резюмує відомості про клієнта операційної системи, такі як бібліотеки, служби запуску, управління пам'яттю, файлова система, серед інших, що забезпечує набагато простіший і практичніший спосіб завантаження та масштабування програм.

Кожен постачальник дає назву цьому типу середовища виконання. Наприклад, в Amazon це називається екземпляр, на OpenShift, на gear і на Heroku у нас є динос. Dynos визначаються як легкі контейнери, які дозволяють розробнику запускати свою програму в ізольованому та безпечному середовищі [16].

Програми, створені для запуску на Heroku, можуть складатися з декількох dyno. Відповідно до їх завдання, dyno можна класифікувати на три типи:

- Web Dynos: відповідає за отримання та зустріч із веб-запитами. Кожна програма може мати не більше одного dyno цього типу;
- Worker Dynos: виконує завдання у фоновому режимі, рекомендуючи їх для більш важкої обробки. Кожна програма може мати кілька dyno такого типу;
- Одноразові Dynos: Вони створені лише для випадкових завдань і незабаром руйнуються. Класичний приклад - читання журналів програм.

2.9. Progressive Web Application

PWA - це гібрид звичайного веб-сайту та мобільного додатку. Уявіть, що коли ви заходите на веб-сайт, який вам дійсно подобається на своєму смартфоні, ви отримуєте можлиість додати веб-сайт на домашню сторінку телефона.

Якщо додаток зараз встановлено на вашому телефоні, ви можете використовувати його так само, ніби ви використовуєте веб-переглядач, без будь-якої зайової інформації на екрані, окрім програми, тобто, всі інтерфейси браузера, такі як адресний рядок та браузерні кнопки будуть видалені.

Прогресивні веб-програми приносять багато інструментів для досягнення нових потенційних користувачів. Так, коли хтось хоче поділитися додатком або навіть певною сторінкою в додатку, можна просто надіслати URL-адресу, а інша людина може легко відкрити PWA, не встановлюючи або надсилаючи на загальну цільову сторінку [10].

Однією з головних особливостей прогресивного веб-додатка є його здатність функціонувати, коли інтернет-зв'язок не стабільний. PWA запускає скрипт у фоновому режимі під назвою Service Worker, мета цього сценарію включає локальний кеш вмісту, що дозволяє програмам завантажуватися практично миттєво. Крім того, якщо користувач переходить у режим офлайн, програма може відображати персоналізований екран, щоб повідомляти про відсутність підключення або, в деяких випадках, навіть доставляти раніше кешований вміст. Після кешування сценарію Service Worker він не тільки має можливість зберігати вміст для використання в режимі офлайн, але і запускати фонові синхронізації для завантаження нових даних. Потім, коли користувач має стабільне з'єднання з Інтернетом, браузер може використовуватися для збору та зберігання додаткової інформації.

Не слід забувати про багатоплатформність та прогресивність. Як правило, прогресивні веб-програми розвиваються відповідно до того, як користувач взаємодіє з ними. Таким чином, це означає, що якщо користувач перегляне програму за допомогою Chrome, Safari або будь-якого іншого браузера, PWA буде постійно вдосконалюватися, а крім того, він також зможе скористатися функціями, доступними на пристрої та браузері користувача. Щоб кинути виклик ідеальному дизайну власних додатків, Progressive Web Apps також надзвичайно добре регаують на зміну платформи. Це означає, що макет програми буде адаптуватися залежно від формату (мобільного, планшетного чи настільного) та розміру екрана, який може сильно відрізнятися.

Ще один вирішальний фактор для поліпшеного функціонування додатку на PWA архітектурі – використання моделі App Shell (Application Shell). Ця структура мінімізує запити на оновлення сторінок, оскільки вона відокремлює частини програми, які часто є постійними, такі як меню, заголовок і макет сторінки, від частин, які регулярно оновлюються. Практично кажучи, прогресивні веб-програми імітують навігацію по мобільних додатках, пропонуючи плавні взаємодії. Це означає, що ключові елементи програми відображатимуться негайно, оскільки вони вже кешовані, покращуючи продуктивність, а потім решта вмісту буде завантажена під час використання.

Мабуть, одна з найбільших переваг перед SPA додатками - це основна функція відкриття Progressive Web Apps. Оскільки вони побудовані на веб-сторінках, можна використовувати потужність пошукових систем для індексації та розподілу програми через результати пошуку. З точки зору маркетингу, це також означає, що деякі методи оптимізації пошукових систем (SEO) тепер можуть застосовуватися до PWA додатків [12].

Веб-безпека - одна з найважливіших тем сучасного життя, і оскільки прогресивні веб-програми базуються на веб-сторінках, є деякі вимоги, яким вони повинні відповідати. Для задоволення вимог безпеки, необхідних для додатків, розробники повинні відображати PWA через захищений HTTP або HTTPS, що дозволяє шифрувати транзит даних. Навіть якщо інформація на веб-сайті не є конфіденційною, конфіденційність та безпека користувача є головними проблемами. Більше того, з новими специфікаціями, зробленими Apple, для додатків iOS та таких служб, як онлайн-платежі, які вже потребують підключення HTTPS, реалізація цієї функції починає набувати головного аспекту.

Деякі можуть вважати негативним аспект процесу встановлення на власних пристроях PWA додатку, оскільки це потребує більшої участі користувачів. Прогресивні веб-програми тепер обходять цей процес і роблять його набагато простішим, з можливістю користувачів додавати додаток на свій

головний екран смартфона або планшета. За допомогою веб-програми розробники можуть подавати браузери з "інсталяційним пакетом", який включає в себе графічні функції, а також інформацію про те, як програма повинна вести себе при запуску. Після того, як додаток встановлено, він набуває статусу вищого рівня, покращуючи користувацький досвід.

Ще одна ключова особливість Progressive Web Apps - це її функції. Раніше доступні лише для нативних програм функці, наприклад поштові нагадування, тепер розробники можуть використовувати і в PWA, використовуючи Service Worker. Наразі PWA push-сповіщення підтримуються Chrome, Firefox, Opera та Safari на персональних комп'ютерах, але не на мобільномих пристрої. У програмах, які вже реалізували цю функцію, спостерігається збільшення часу на додаток на 72%, а повторні відвідування збільшилися більш ніж на 50% [2].

2.10. Single Page Application

SPA-абревіатура розшифровується як програма на одній сторінці, але це не означає, що у додатку буде лише одна сторінка.

Що насправді змінюється, це спосіб завантаження сторінки. Ми звикли до додатків, де сторінки відображаються на стороні сервера, незалежно від використовуваної технології. Це має ефект: кожна нова сторінка, яку потрібно завантажити, перетворюється на новий запит на сервер, цей запит, щоб браузер міг завантажувати HTML, CSS та JavaScript нової запитуваної сторінки. Тому в кількох веб-додатках ми бачимо, що браузер забирає «мало часу» для завантаження нової сторінки, або ми бачимо, що нова сторінка спочатку порожня, щоб завантажуватися наступною.

Програми, засновані на рамках SPA, працюють по-різному, оскільки не потрібно робити запитів на завантаження нових сторінок. Додаток буде завантажений у повному обсязі за першим запитом, де всі необхідні HTML, CSS та JavaScript будуть завантажені одразу. З цього моменту, коли потрібно

буде завантажувати нові сторінки, вони завантажуватимуться через підпрограми JavaScript, усуваючи потребу в запитах на сервер, щоб отримати новий вміст.

SPA-програми в цілому дозволяють нам отримати деякі переваги. Одна з них - це можливість оптимізувати продуктивність програми в цілому, перекладаючи всі зусилля на сторону клієнта та дозволяючи більш легкий трафік даних між клієнтом та сервером. Іншою перевагою є повторне використання коду за допомогою таких технологій, як React / React Native та Angular / Ionic, що дозволяє розробляти систему з меншими зусиллями та більш стандартизовано навіть для мобільних додатків. Однак у SPA-додатках, як правило, є деякі слабкі місця, такі як render тільки на стороні клієнта, а в деяких випадках і проблеми з SEO.

В додатку SPA завантаження ресурсів (таких як CSS, JavaScript та HTML сторінок) відбувається лише один раз - коли перший раз користувач відкриває сайт. У цьому першому доступі весь вміст HTML, CSS та JavaScript вже передається клієнту. З цього моменту, коли користувач переміщається через сторінки додатків, більше не потрібно буде робити запити на сервер для завантаження нових сторінок: вміст, пов’язаний з ними, вже завантажений при першому доступі. У подальшому вміст сторінки завантажується через JavaScript код, який генерується на основі фреймворків SPA, таких як Angular, React та Vue.js. Тому ми говоримо, що обробка завантаження сторінок та відповідних їх ресурсів переходить до клієнта.

На даний момент сервер більше не відповідає за візуалізацію вмісту, а за обробку даних, якими оперує додаток. Операції з доступом до баз даних або повернення контенту, що підлягає відтворенню, стають функціями розміщеними виключно на стороні сервера. Наразі загальним є те, що сервер виставляє API RESTful для його використанням додатку SPA. Програма SPA спілкується з цим RESTful API за допомогою HTTP-запитів, передаючи дані у форматах XML та JSON. Додаток на стороні сервера відповідає за реагування

на ці HTTP-запити, виконуючи бізнес логіку необхідну конкретному запиту. Ця архітектура має дуже очевидну перевагу: повне розділення та ізоляція серврної та клієнтської частини. Це дає можливість двом командам паралельно працювати не заважаючи один одному, наприклад, тепер є можливість направити людей які більш знайомі з роботую серверної частини виконувати задачі пов'язані тільки з нею. Так само і з клієнтською частиною. Це дозволить більш цілеспрямовано розподіляти ресурси людей.

Наразі SPA-програми можна використовувати практично у будь-яких ситуаціях, що пояснює популяризацію SPA-систем, таких як Angular, React, Vue.js та Ember сьогодні. Однак є деякі ситуації, коли рамки SPA можуть бути не такими придатними.

Програми, які потребують надзвичайного SEO, можуть мати проблеми, якщо вони розроблені за допомогою SPA-систем. Більшість індексаторів сьогодні здатні зрозуміти SPA-додатки, крім того, що існує велика кількість метатегів для покращення цієї точки. Однак це додаткова робота порівняно з класичним візуалізацією декількох сторінок на сервері. Ця "маленька проблема" трапляється саме тому, що вміст не виводиться на сервер, а повністю видається клієнту. З цієї причини пошуковим системам може бути важко індексувати вміст сторінок, оскільки вони не можуть індексувати вміст, створений на стороні клієнта. Це пояснює, чому сьогодні більшість фреймворків SPA мають рішення SSR (рендер сторінок на стороні сервера), де принаймні частина вмісту завантажується на стороні сервера, щоб сприяти механізмам індексації [11].

SPA-програми можуть бути трохи повільнішими, особливо при першому завантаженні. Це тому, що весь вміст потрібно завантажити відразу клієнту під час першого доступу. Крім того, оскільки весь процес візуалізації стає відповідальністю клієнта, додаток трохи постраждає через можливі апаратними обмеженнями тих, хто має використовує його. Однак, очевидно, є прийоми для зменшення цієї точки, наприклад, ліниве навантаження;

Взагалі для програм SPA потрібен JavaScript у браузері, щоб його було включено, хоча сьогодні це не так часто, щоб його відключити. Існують також методи, які можуть пом'якшити цю точку, наприклад, SSR.

Існує незначна тенденція, щоб SPA-програми були менш захищеними, ніж багатосторінкові програми, що надаються на сервері, особливо стосовно XSS-атак. Однак це не правда і насправді більше пов’язано з знаннями розробника про методи запобігання подібним видам атак. Запобіжні заходів, слід дотримуватися навіть у додатках з рендером на стороні сервера.

Увага до крихкості екосистеми JavaScript - це дуже суперечливий момент, але факт, що «нестабільність» фреймворків JavaScript необхідно враховувати. Починаючи розробляти додаток використовується певна версія SPA фреймворку але при оновленні версії є потреба в переписанні значної кількість коду, спочатку використаного в проекті. Це, без сумніву, важливий фактор, який слід враховувати не тільки, коли мова йде про використання SPA-бази або рамки MPA, але і коли йдеться про рішення, яку структуру SPA використовувати.

2.11. PostgreSQL

Одне з важливих питань при розробці будь-якої програми, яким необхідно зберігати інформацію в базі даних - це вибір системи управління базами даних або СУБД. Хоча ми можемо керувати збереженими даними альтернативними способами, СУБД є найбільш ефективною, а також найбільш сумісною з технологіями, що застосовуються на ринку.

Серед найбільш відомих систем PostgreSQL - це система управління реляційними базами даних, що виділяються своїм відкритим кодом та тим, що є одними з найсучасніших з точки зору ресурсів у цьому сегменті.

PostgreSQL - це проект, який спочатку розроблявся за ліцензією BSD і отримав свою першу версію у 1989 році. Основна мета створення PostgreSQL полягала у створенні СУБД, яка розуміла різні типи даних і яка могла описати зв’язки між ними [6].

У цій версії система підтримувала мову QUEL, яка була замінена мовою на SQL через 5 років. В даний час розробка ведеться за моделлю Базара [4], в якій розробляється код абсолютно відкритим і публічним способом. Його розробники - це переважно волонтери у всьому світі, які спілкуються через Інтернет. Перевірка відбувається групою людей, спеціально відібраних для цієї мети. PostgreSQL фінансується такими великими компаніями, як Fujitsu, Hub.Org, NTT Group, Red Hat, Skype та SRA. Його користувачами є транснаціональні організації, урядові установи різних країн та великі університети. PostgreSQL.

Якщо порівнювати його конкурента з відкритим кодом MySql, то перевага останнього полягає в його простоті для освоєння. Причину цього можна пояснити тим, що PostgreSQL більш надійний і має більше функцій, що вимагає більш глибоких технічних знань. Однак, для виправлення цього фактору був проведений значний обсяг роботи що в результаті майже нівелювало різницю між цими конкурентами. Ще одним важливим фактом, що перешкоджав популяризації PostgreSQL, є відсутність версії СУБД для середовища Windows, що було виправлено лише у восьмій версії [3].

В результаті PostgreeSQL стала реляційною системою управління що має високу продуктивність, легке адміністрування та використання в проектах. Можливість писати на SQL, використовувати тригер та багато інших ресурсів, присутніх у найвідоміших СУБД на ринку, таких як Oracle, InterBase, SQL Server, MySQL. Крім того, PostgreSQL має інтерфейси для взаємодії з різноманітними середовищами та мовами програмування. Усі ці функції надають програмістам та адміністраторам баз даних можливість виконувати поставлені задачі швидше та зручніше. Однією з його головних характеристик PostgreSQL є той факт, що це відкрите програмним забезпеченням, тобто його використання, модифікація чи розповсюдження є безкоштовними. З моменту як PostgreSQL став «open source» програмним забезпеченням в середині 1996

року, було зроблено багато змін, що зробили його сильним конкурентом найсучасніших СУБД на ринку.

2.11. Swagger

Документування коду - важливий момент будь-якого проекту, який часто не помічають. Працюючи в команді, погана документація може ускладнити роботу інших розробників.

Swagger - це програма з відкритим кодом, яка допомагає розробникам визначати, створювати, документувати та використовувати REST API. Також він надає інструменти для: допомоги у визначенні конфігурації (редактор Swagger), взаємодії з API через інтерфейс Swagger (рис 2.2) та генерування шаблонів коду з файлу конфігурації (Swagger Codegen)

Він складається з файлу конфігурації, який можна визначити в YAML або JSON.

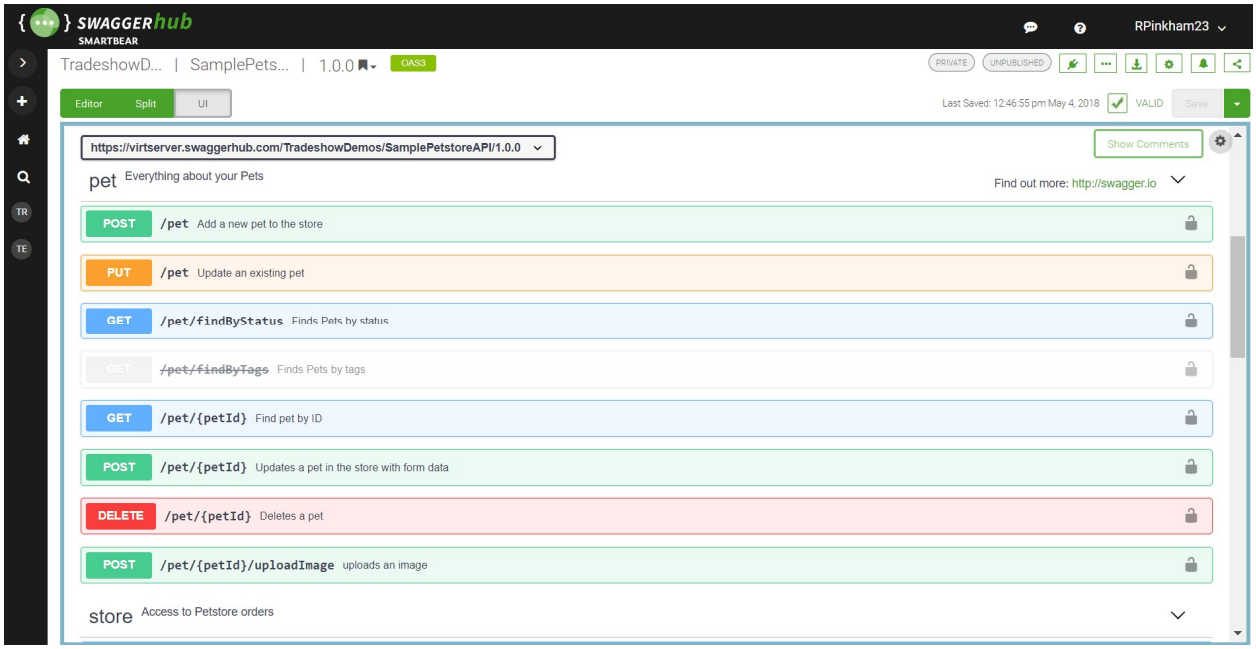


Рис 2.2. Інтерфейс Swagger

Як бачите, найважливішим моментом є файл конфігурації Swagger. Тут документується API.

2.12. WebSocket

Мережа була побудована на основі відомої парадигми HTTP яка полягає у тому що на один запит отримується одна відповідь. Клієнт завантажує веб-сторінку, і нічого не відбувається до тих пір, поки користувач не натисне на кнопку що завантажить наступну сторінку. Близько 2005 року модель AJAX почала робити Інтернет більш динамічним. Незважаючи на це, весь HTTP-зв'язок керував клієнтом, який вимагав взаємодії користувача або періодичного опитування для завантаження нових даних із сервера.

Технології, які дозволяють серверу надсилати дані клієнтові одночасно, коли він виявляє, що нові дані доступні, використовуються вже деякий час. Вони були відомі такими іменами, як "Push" або "Comet". Одна з найпоширеніших проблем, що створюють ілюзію ініційованого сервером зв'язку, називається тривалим опитуванням. При тривалому опитуванні клієнт відкриває HTTP-з'єднання до сервера, яке залишається відкритим, поки відповідь не буде надіслана. Щоразу, коли у нього з’являються нові дані, сервер надсилає відповідь (інші методи включають Flash, багатозахисні запити XHR і так звані htmlfiles). Тривалий опитування та інші методи дуже добре працюють. Однак усі ці рішення поділяють проблему: вони несуть велике навантаження на мережу та не підходить для програм які вимагають низьку затримку між запитом та відповіддю сервера. Подумайте про багатокористувацькі ігри в браузері або будь-яку іншу онлайн-гру з компонентом у режимі реального часу.

Специфікація WebSocket визначає API, який встановлює "сокетні" з'єднання між веб-браузером і сервером. Іншими словами, між клієнтом і сервером існує стійкий зв’язок, і обидві сторони можуть розпочати надсилання даних у будь-який час [14].

З'явилась нова схема URL для підключень WebSocket. Існує також wss: для безпечного з'єднання WebSocket і, як і https: він використовується для захищених HTTP-з'єднань.

Кожна нова технологія має новий набір проблем. Що стосується WebSocket, саме сумісність із проксі-серверами опосередковує HTTP-з'єднання у більшості корпоративних мереж. Протокол WebSocket використовує систему оновлення HTTP (яка зазвичай використовується для HTTP / SSL) для "оновлення" HTTP-з'єднання до з'єднання WebSocket. Деяким проксі-серверам це не подобається, і вони відмовляться від з'єднання. Таким чином, навіть якщо певний клієнт використовує протокол WebSocket, встановити з'єднання може не вдасться.

Однак ви можете використовувати WebSocket сьогодні з бібліотеками, які беруть на себе ситуації, коли WebSocket недоступний. Бібліотека, що стала дуже популярною в цьому домені, - socket.io, яка постачається з реалізацією протоколу клієнтом і сервером. Існують також комерційні рішення на зразок PusherApp, які можна легко інтегрувати у будь-яке веб-середовище, надаючи HTTP API для надсилання повідомлень WebSocket клієнтам. Завдяки додатковому запиту HTTP, завжди будуть додаткові витрати порівняно з чистими WebSocket.

WebSocket створює абсолютно новий шаблон користування для серверних додатків. Хоча традиційні стеки серверів на зразок LAMP розроблені на основі циклу запитів / відповідей HTTP, вони, як правило, не справляються з великою кількістю відкритих підключень WebSocket. Утримування великої кількості з'єднань відкритим одночасно вимагає архітектури, яка отримує високу конкуренцію з низькою продуктивністю. Ці архітектури, як правило, розробляються на основі ланцюга або так званого IO без блокування.

Останні версії Google Chrome та Google Chrome для Android повністю сумісні з протоколом рукостискання та передача даних між клієнтом та сервером через WebSocket, включаючи двійкові повідомлення [15].

Використовуйте WebSocket дуже зручно, коли вам потрібна низька затримка та підключення в реальному часі між клієнтом і сервером, наприклад багатокористувацькі онлайн ігри, online-чати, соціальні мережі.

ВИСНОВОК ДО РОЗДІЛУ 2

В даному розділі було детально досліджено технології необхідні для створення web-додатків. Також було розглянуто декілька видів архітектурних підходів до реалізації веб-порталу.

Спочатку були вибрані основні інструменти у виді мов програмування та фреймворків. Вибір був зосереджений на мовах програмування що постійно розвиваються та мають певний вплив на сучасному ринку. Тому для серверної частини була обрана Java. Вона постійно розвивається та є дуже популярною при створенні веб-додатків. Другою мовою був вибраний TypeScript – мова програмування, що побудована на JavScript але має чудову підтримку об'єктно орієнтованого підходу.

Другим кроком були вибрані фреймворки на основі яких буде реалізована система. Непоганою парою для Java став Spring Boot, який за багато років завоював довіру великої кількості розробників своєю надійністю на зручністю. Ще одним фреймворком, уже для клієнтської частини, був вибраний Angular. Основним при його виборі стала підтримка сучасних архітектурних підходів та велика аудиторія інших користувачів що полегшує розробку додатку. Після дослідження та аналізу можливих підходів до реалізації «портальної» частини системи був вибраний SPA підхід.

Для розгортання системи використовуватиметься хмарний сервіс Heroku який чудово підходить для додатків написаних на Java та баз даних PostgreSQL.

Також були досліджені технології без яких розробка веб додатку буде досить складною, а саме: HTML, CSS, TypeScript, Apaceh Tomcat, Swagger, Web Socket.

Всі ці технології допоможуть реалізувати поставлену мету, і створити систему, яка буде працювати з великою кількістю користувачів одночасно.

РОЗДІЛ 3.
ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ДОДАТКУ

3.1. Проектування основних компонентів системи

З огляду на орієнтованість системи при розробці системи було створено декілька видів користувачів (рис 3.1.) розподіливши їм необхідні ролі в системі.

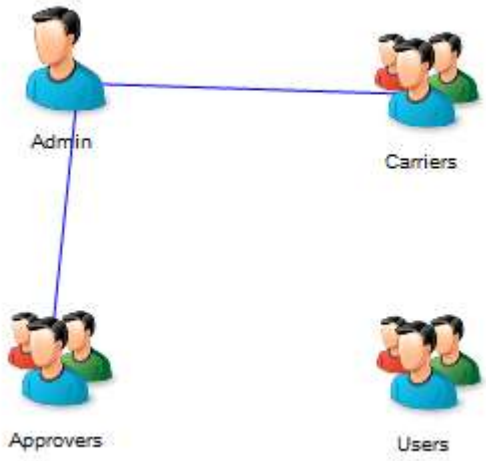


Рис 3.1. Види користувачів в системі

Адміністратор в системі існує тільки один і в його можливості входять такі функції:

- створення користувачів з роллю провайдера;
- створення інших користувачів з роллю редактора;
- можливість виконувати ті ж функції що і користувачі з роллю редактора.

Редакторів в системі може бути безліч. Вони виконують такі функції як:

- перевірка створених подорожей на валідність;
- вирішення питань при створенні подорожі у провайдера;
- вирішення питань звичайних користувачів які виникли під час користування наданими провайдером послугами.

Провайдерів в системі також може бути необмежена кількість адже можливість інших компаній продавати свої послуги є однією з поставлених цілей. Як зрозуміло з назви, провайдер має можливість виконувати такі функції як:

- додавати в систему нові подорожі та редагувати існуючі;
- додавати до подорожей сервіси;
- створювати акції для своїх сервісів.

Роль звичайного користувача не потребує детального огляду, так як не передбачає жодних незвичних в наш час функцій.

Для того щоб зробити розробку системи більш ефективною, на основі раніше сформульованих критеріїв, було створено найбільш поширений сценарій користування системою, схема якого зображена на рисунку 3.2.

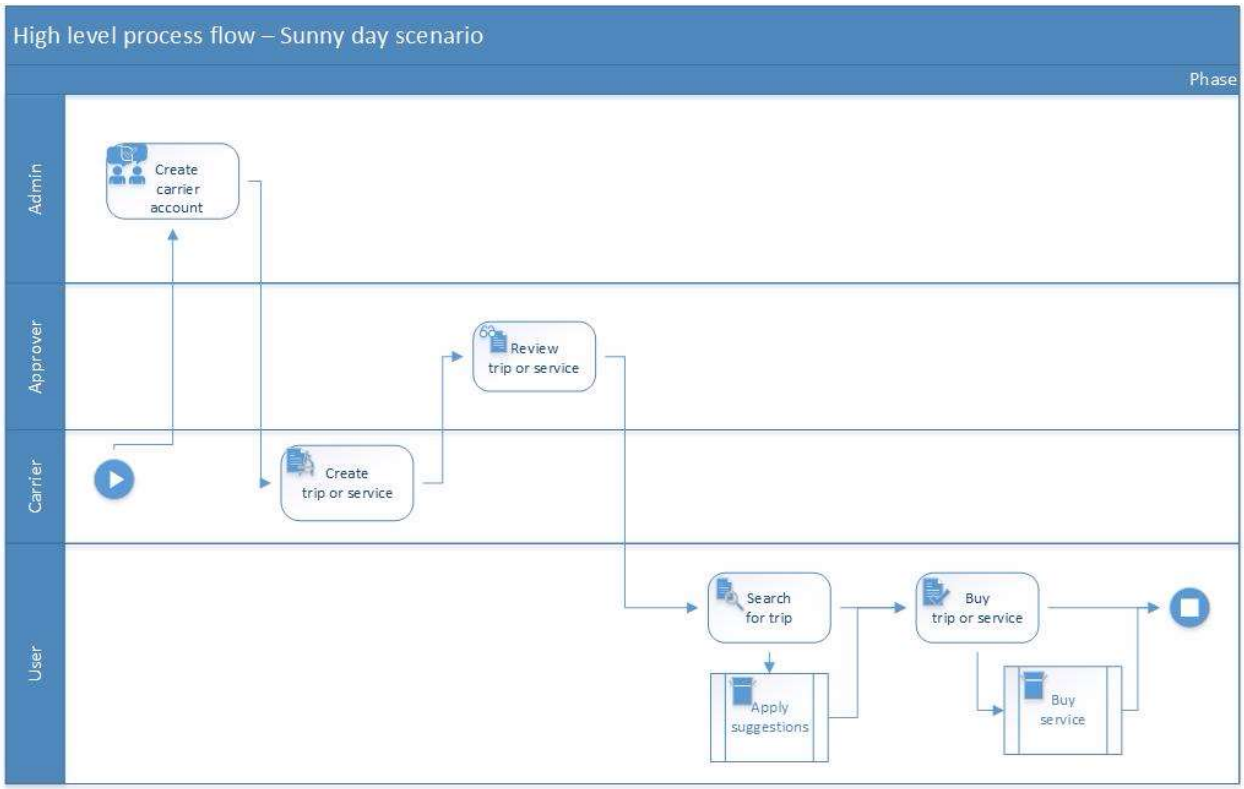


Рис. 3.2. Основний сценарій користування системою.

Особливої уваги при розробці системи вимагала схема життєвого циклу подорожі (Рис. 3.3) та «трабл тікетів» (Рис. 3.4.)

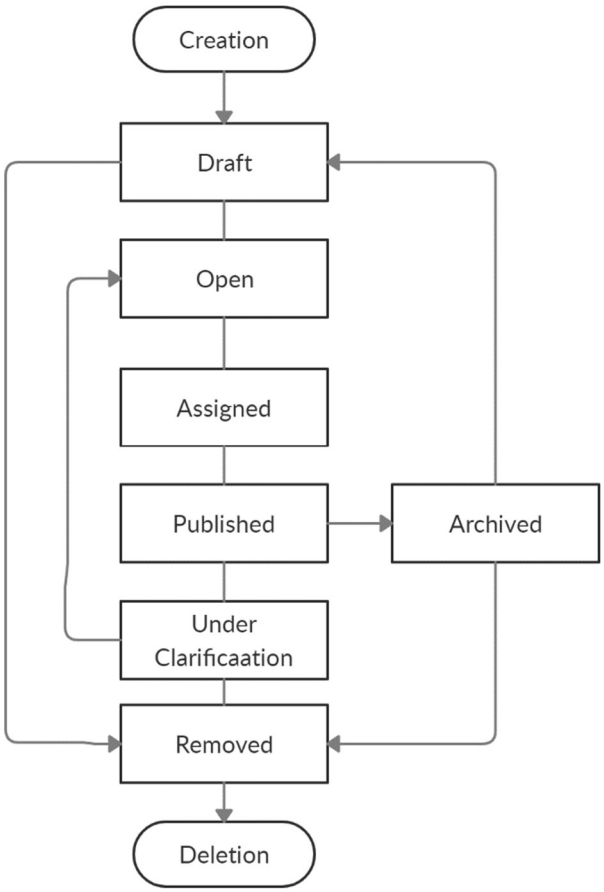


Рис. 3.3. Життєвий цикл подорожі

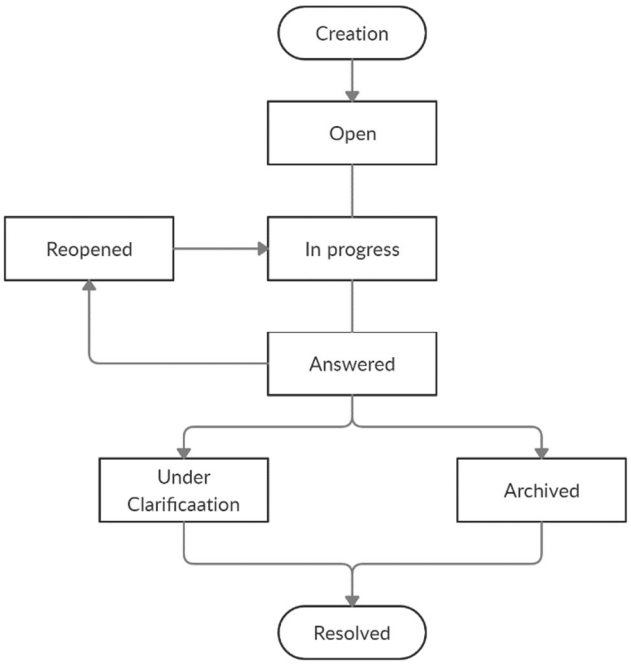


Рис. 3.4. Життєвий цикл скарги (trouble ticket)

3.2. Розробка серверної частини системи

Для реалізації схожих елементів системо доцільно використовувати інтерфейси та реалізовувати відповідну кожному сервісу логіку окремо. Тому було створено інтерфейс клас GenericDao в якого закладені пристні для більшості сервісів методи, а саме:

- List<T> getAll() - в цьому методі реалізовано всю логіку по отриманню всіх елементів у виді списку;
- Optional<T> getById(Long id) – метод для отримання об’єкту по його айді;
- T insert(T entity) – метод яким можна додати об’єкт в базу даних;
- void update(T entity) – метод який надає можливість змінити поля в уже існуючому об’єкті;
- void delete(T entity) – метод для видалення об’єкту з бази даних;
- void deleteById(Long id) - метод для видалення об’єкту з бази даних по його айді;
- Optional<T> getSingleElement(List<T> element) – метод, який повертає список елементів розмір якого завжди рівний одинці.

Всі сервіси повинні реалізовувати такий список методів, і таким чином, надавати відповідну функціональність при користуванні ними. Але можливостей що надають вище перераховані методи дуже часто не вистачає, тому було створено методи для реалізації більш специфічних потреб.

Для прикладу розглянемо ServiceDao інтерфейс та його методи:

- Optional<Service> getTripByIdDetailed(long id) – використовується для отримання детальної інформації про подорож;
- List<Service> getSuggestionsByTripId(long id) – необіхдний для отримання сервісів які наявні для конкретної туристичної поїздки;
- List<Service> getAllTripsOfStatus(String status) – повертає поїздки з заданим статусом;

- List<Service> getTripsByProviderId(String serviceTypeName, long id) – повертає поїдки конкретного перевізника;
- List<Service> getTripsOfStatusByProviderId(String status, long id) - повертає поїдки конкретного перевізника з указаним статусом;
- List<Service> getTripsByApproverId(long id) – повертає поїздки які обробляються конкретним редактором;
- List<CountryTrips> getTripsAmountByCountry() – повертає кількість поїздком у кокретній країні;
- List<Service> getAllBundles() – повертає всі бандли;
- List<Service> getAllBundlesDetailed() – повертає всі бандли та їх детальний опис;
- Optional<Long> countAllTrips() – повертає кількість всіх поїздок;
- Optional<Long> countAllServices() – повертає кількість всіх сервісів;
- Optional<Long> countAllBundles() - повертає кількість всіх бандлів;
- List<Service> getTripsByProviderIdWithLocationAndStatus(long id) – повертає поїздки конкретного провайдера відфільтровані по локаії та статусу;
- List<Service> getPublishedWithLocationsAndStatus(String serviceType) – повертає опцубліковані поїздки;
- void setRemovedForAllUnderClarification() – позначає всі поїздки з статусом «Under Clarificartion» як видалені;
- List<Service> getAllServicesPurchasedByUserWithUserId(long id) – знаходить всі поїздки які придбав користувач;
- Optional<Long> getSearchByNumOfOrderProviderId(long userId) – повертає кількість створених для кожного провайдера;
- List<Service> getSpecialSearch(String request, Object[] objects) – пошук поїздок по спеціальним критеріям;
- Optional<Long> countPublishedServices() – повертає кількість опублікованих поїздок;

- Optional<Long> countUnPublishedServices() – повертає кількість неопублікованих поїздок.

3.3. Проектування та розробка бази даних

Для того щоб створити базу даних спочатку необхідно придумати основні сутності що будуть існувати в системі та створити для них відповідні таблиці. Зв’язавши створені сутності декількома типами зв’язків та зобразивши це ми отримаємо ER-діаграму бази даних. Діаграму яка відображає спроектовану базу даних можна побачити на додатку 2.

В процесі проектування бази даних було створено 29 таблиць, зупинимося на декількох більш детально.

User – таблиця, в якій зберігається інформація про користувача, вона містить п’ять полів:

- id – унікальний ідентифікатор користувача;
- authorityId – ідентифікатор ролі користувача (foreign key з полем id таблиці Authority);
- email – електронна адреса користувача;
- username – псевдонім користувача;
- password – поле в якому зберігається хеш пароля користувача.

Ця таблиця розширяється таблицею UserDetails і яка містить такі поля:

id – унікальний ідентифікатор користувача (відповідно id з таблиці User);

firstName – ім’я користувача;

lastName – прізвище користувача;

registrationDate – дата коли був створений користувач;

locationId – айді місця проживання користувача;

imageSrc – посилання на сховище де зберігаєтсья автар користувача.

За схожим принципом створені і інші таблиці в базі даних що дозволяє продовжити розробку програмного продукту.

3.4. Розробка клієнта для веб-додатку

Однією серед поставлених задач була реалізація функції пошуку поїздок. Таким чином на рисунку 3.5 зображено поля з фільтрами за допомогою яких можна шукати подорожі. Пошук здійснюється за рахунок пошуку в глибину (Дод. 3) по графу маршрутів та постфільтрації по іншим параметрам.

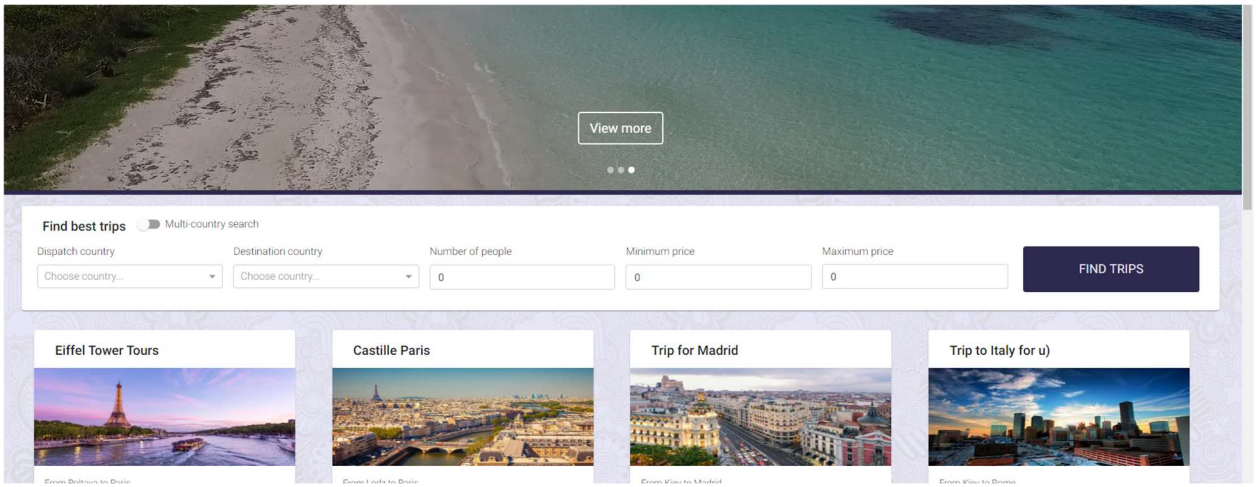


Рисунок 3.5 Фільтри для пошуку подорожей

Через специфіку деяких функцій які можливо використовувати на реалізованій системі реєстрація є ключом до розблокування повних можливостей сервісу. Користувачу який має акаунт в системі надається можливість здійснювати покупки, використовувати чат для спілкування з іншими користувачами, залишати коментарі.

Авторизованому користувачу стає доступний особистий кабінет де він може переглядати та редагувати інформацію про себе, керувати своїми підписками (рис. 3.6).

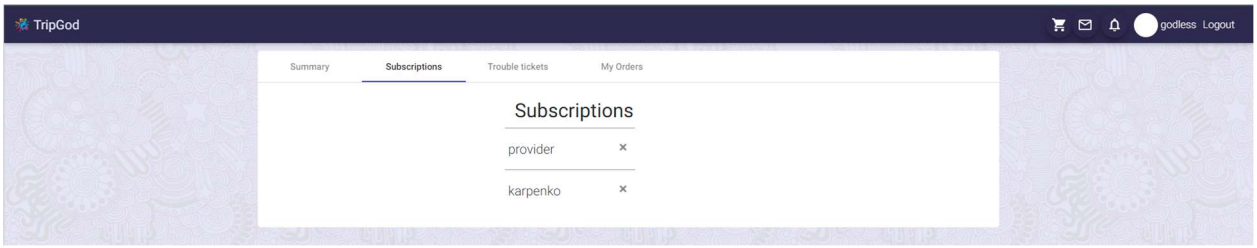


Рис. 3.6 – Сторінка керування підписками

Також ще стає можливим перегляд уже придбаних подорожей та створення скарг у виді «трабл тікетів» (рис. 3.7.)

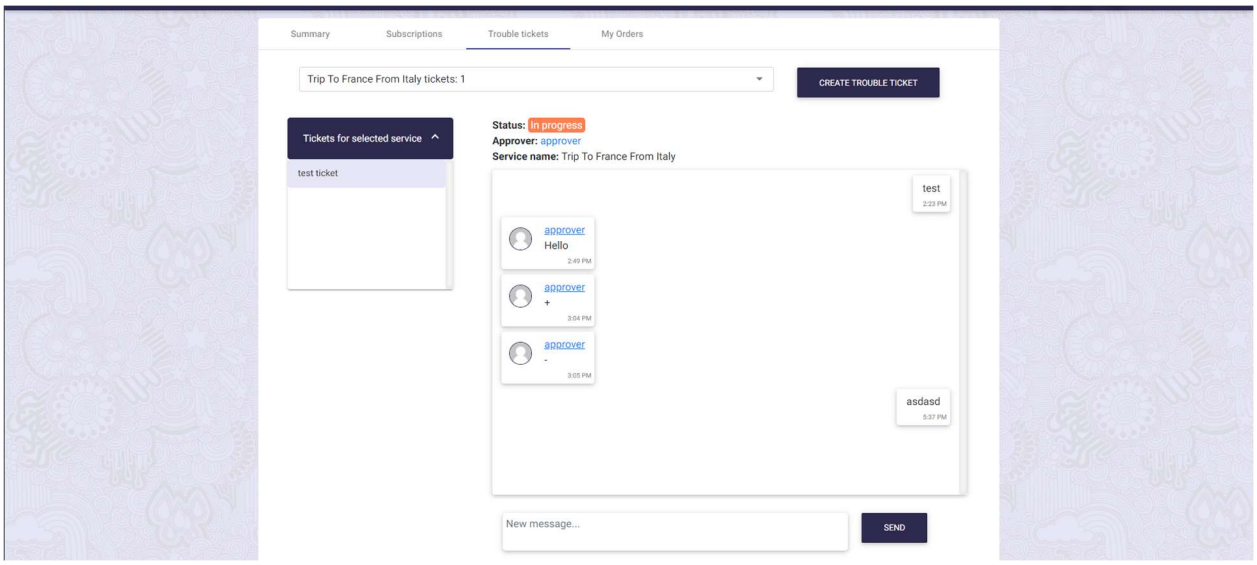


Рис. 3.7. Сторінка «трабл тікетів»

Корзина в якій можна керувати вибраними для покупки подорожами має вигляд спливаючого модального вікна (рис 3.8) в якому є можливість видаляти «товари» або здійснювати їх покупку.

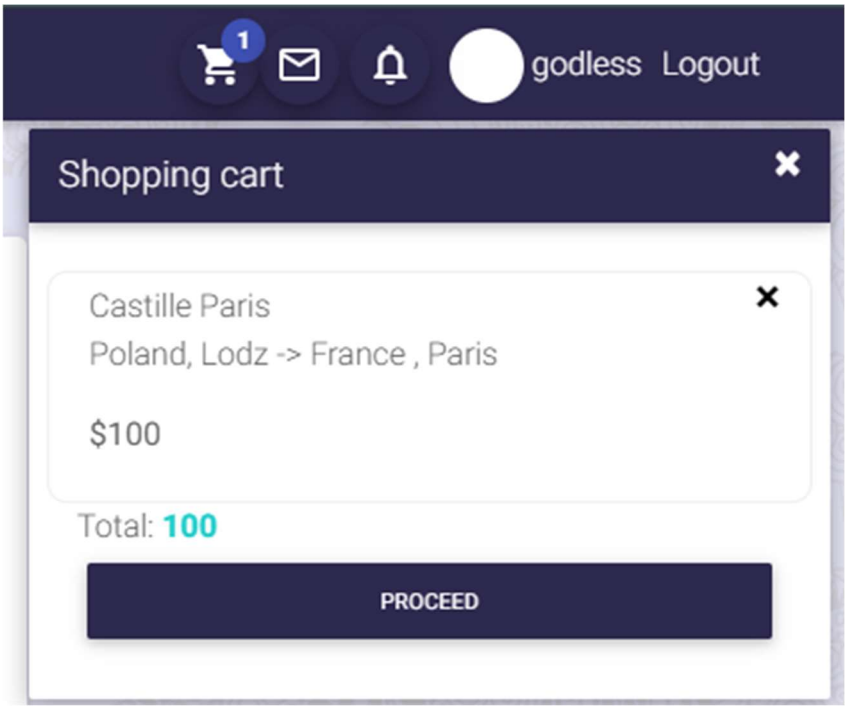


Рис 3.8. Корзина з вибраними «товарами»

В системі реалізована система спілкування у виді онлайн чатів (рис. 3.9.) Їх існує декілька видів:

- глобальний чат – в ньому можуть спілкуватись усі користувачі системи;
- «support» чат – чат, в якому, при необхідності, можна попросити допомоги у адміністраторів сайту;
- «trouble tiket» чат – місце в якому можна зробити скаргу на куплену послугу.

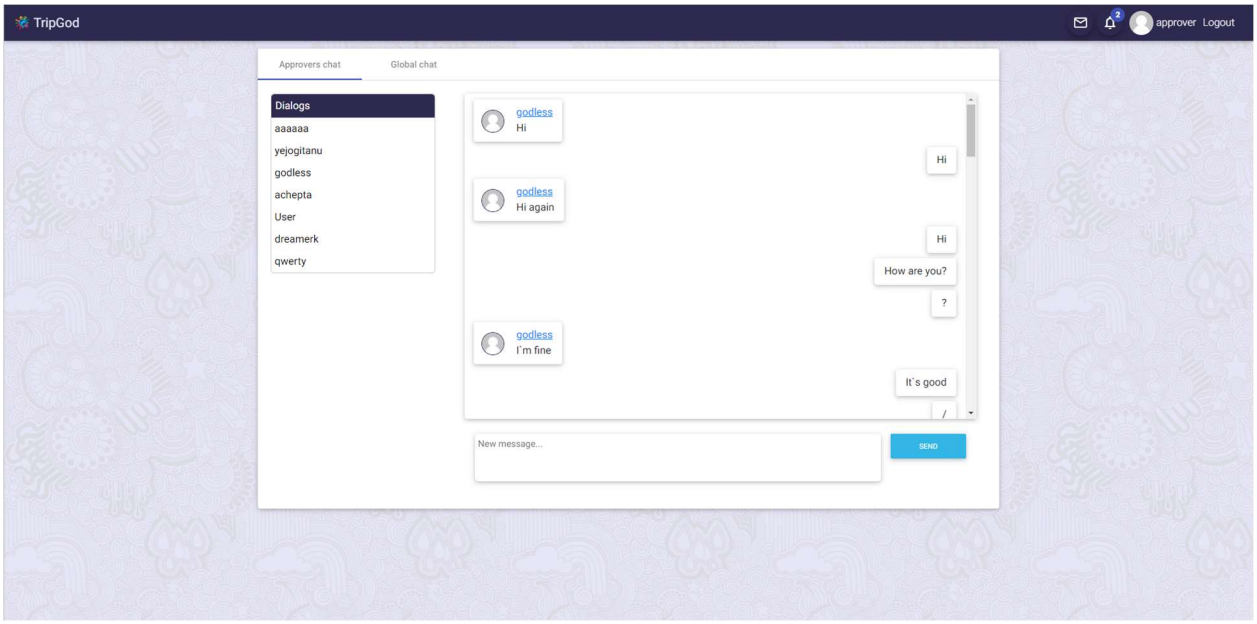


Рис. 3.9. Сторінка з чатами

Залежно від ролі користувакча в системі вигляд деяких чатів може відрізнятись. Наприклад, у адміністратора і редактора системи є можливість відповідати всім користувачам в «support» чатах, а для звичайного користувача є тільки одне діалогове вікно.

Для сповіщення користувачів про важливі зміни в системі реалізовано механіз сповіщень. Сповіщення можна побачити в швидкому доступі – спливаюче модальне вікно з останніми сповіщеннями (рис 3.10), та в менеджері сповіщень, де можна прочитати всі наявні сповіщення (рис 3.11).

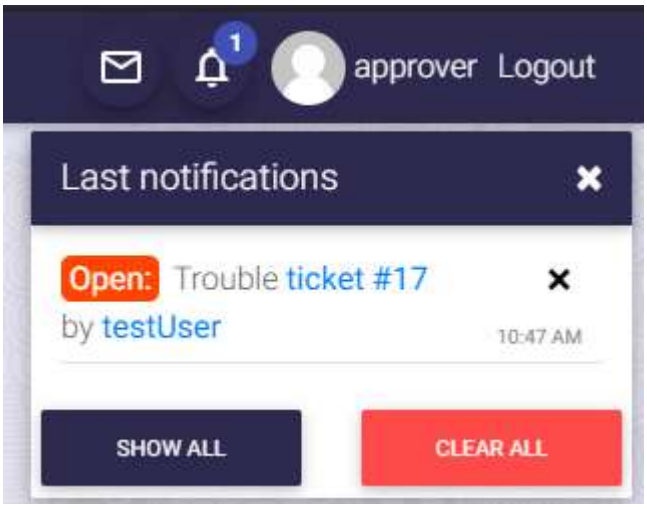


Рис 3.10. Швидкий доступ до сповіщень

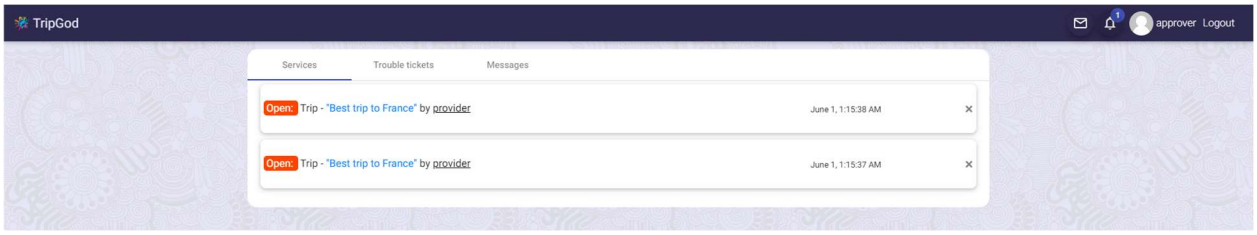


Рис 3.11. Повний список сповіщень

На рисунку 3.12 зображено коментарі користувачів, що уже скористалися наданими провайдером послугами. Крім самого коментаря є також шкала оцінки того, наскільки сподобався сервіс користувачам.

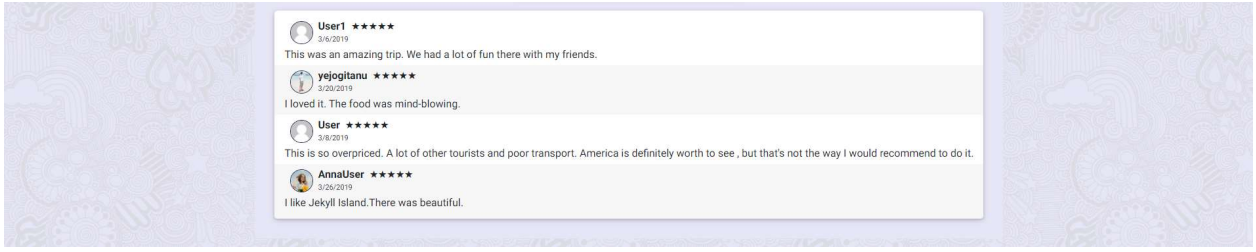


Рис 3.12. Приклад коментарів до подорожі

Також для адміністратора системи збирається статистки про ключові моменти в розвитку веб-сервісу. Для того щоб спростити аналіз зібраних даних будуються графіки (рис. 3.13). Також є можливість експортувати ці дані в формат електронних таблиць.

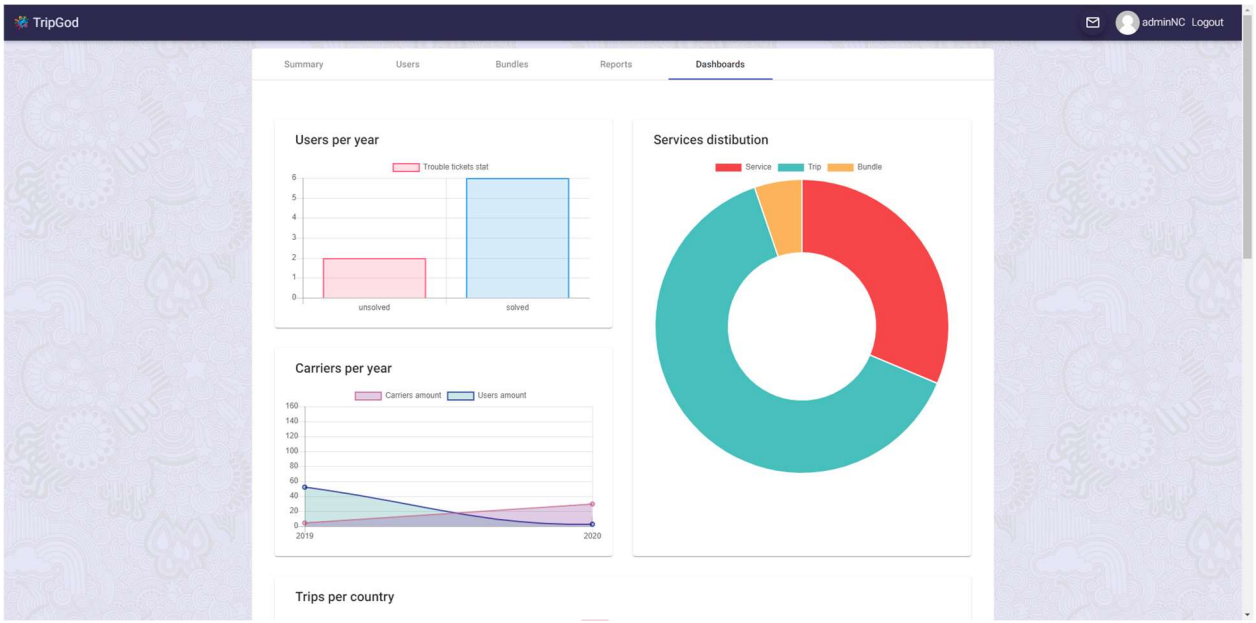


Рис 3.13. Графіки з статистикою веб-сервісу.

ВИСНОВКИ ДО РОЗДІЛУ 3

В даному розділі було описано процес розробки клієнт-серверної системи.

На основі критерії визначених в розділі 1 був спроектований основний сценарій за яким користувачі будуть використовувати систему. Це дало змогу визначити основні елементи необхідні для написання серверної частини та бази даних. Продумана архітектура дозволила швидко та зручно розширяти систему новими функціями.

Не менш важливим став етап розробки клієнта для такої системи адже лише через нього користувачі будуть оцінювати успішність системи.

Реалізовану систему було розміщено на хмарному сервісі Heroku для тестування, в процесі якого була знайдена та виправлена частина недоліків та неполадок.

РОЗДІЛ 4.
ІНСТРУКЦІЯ КОРИСТУВАЧА

1) Спочатку необхідно відкрити веб-сторінку за адресою http://tripgod.herokuapp.com. На рисунку 4.1 зображена початкова сторінка сайту.

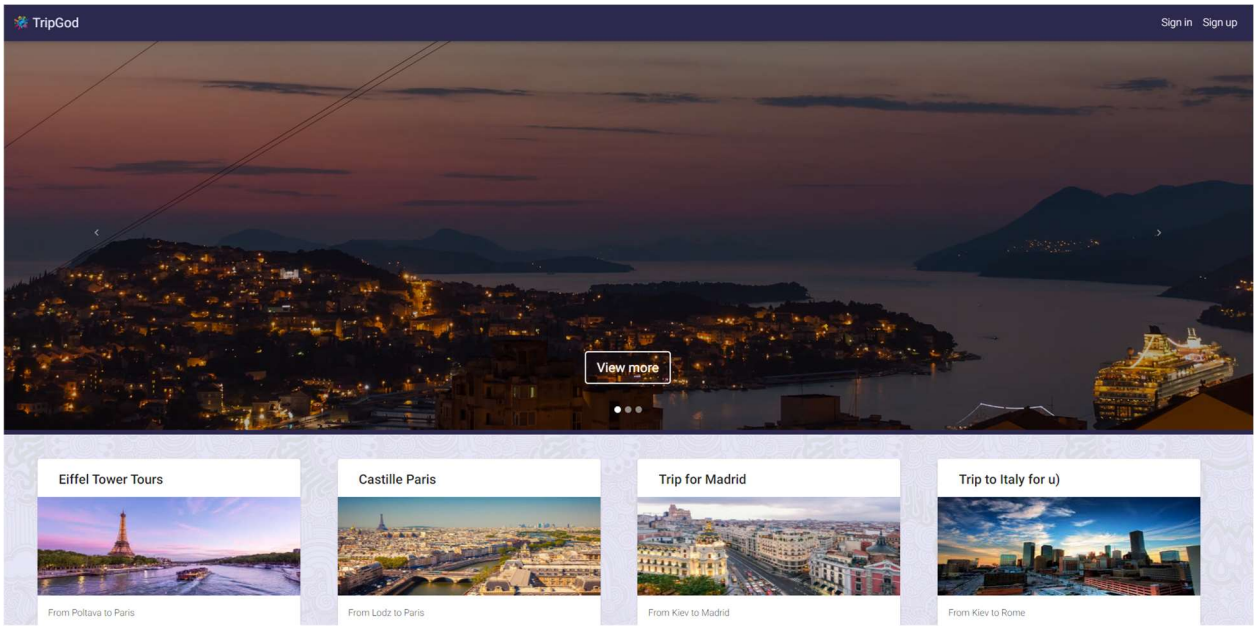


Рис. 4.1. Початкова сторінка веб-сервісу

2) Далі необхідно зареєструватись заповнивши всі необхідні поля як на рис. 4.2.

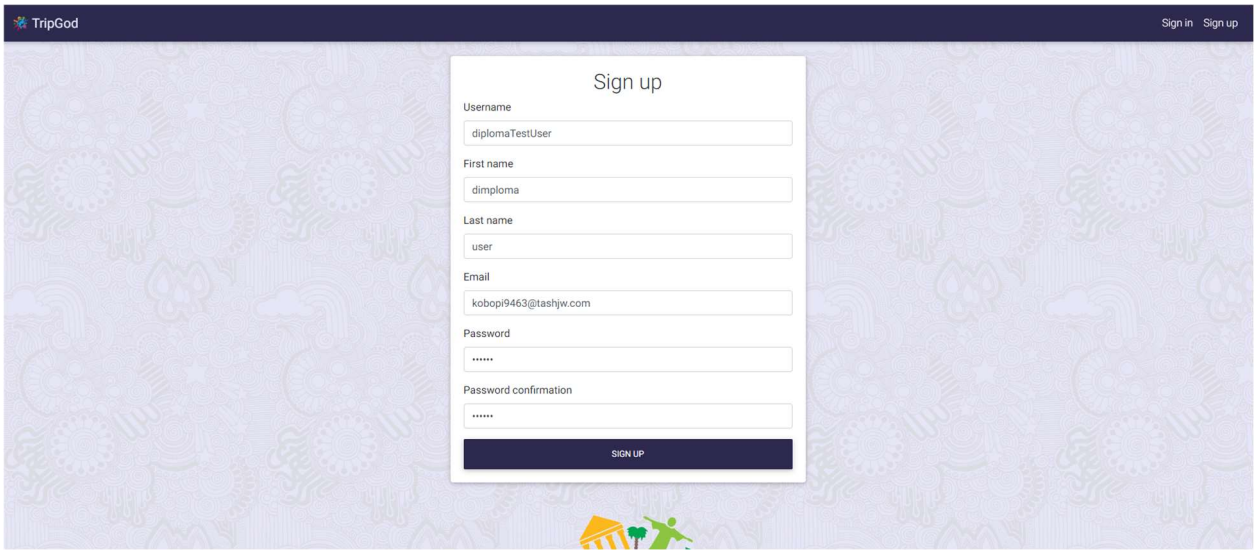


Рис 4.2. Сторінка реєстрації нового користувача

3) На пошту прийде лист з підтвердженням (рис. 4.3).

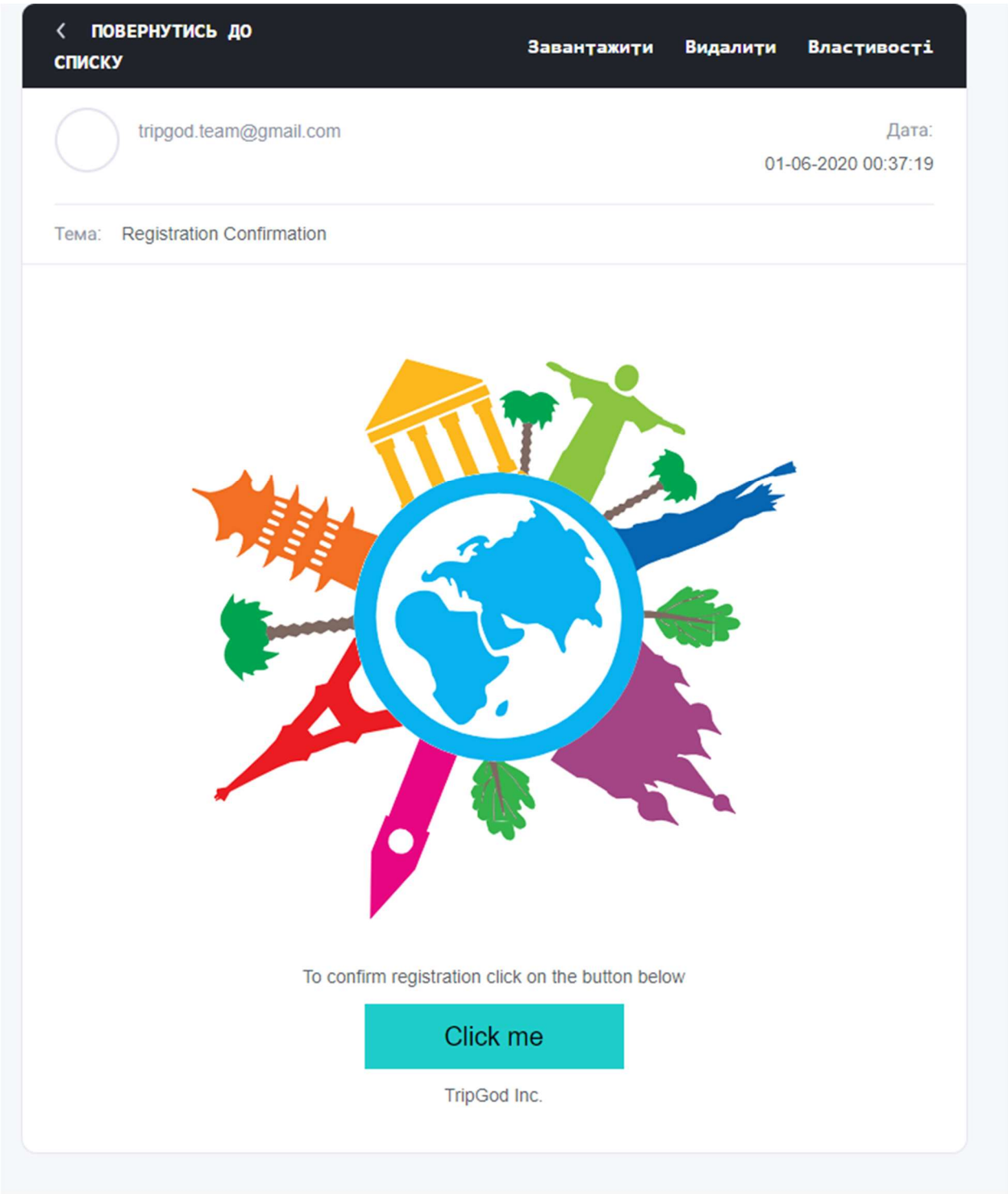


Рис 4.3. Лист з підтвердженням реєстрації

4) Після підтвердження реєстрації у користувача з’являється можливість користуватись такими функціями сервісу як редагування даних (рис 4.4)

в своєму акаунті замовлення подорожей та переписки з іншими користувачами.

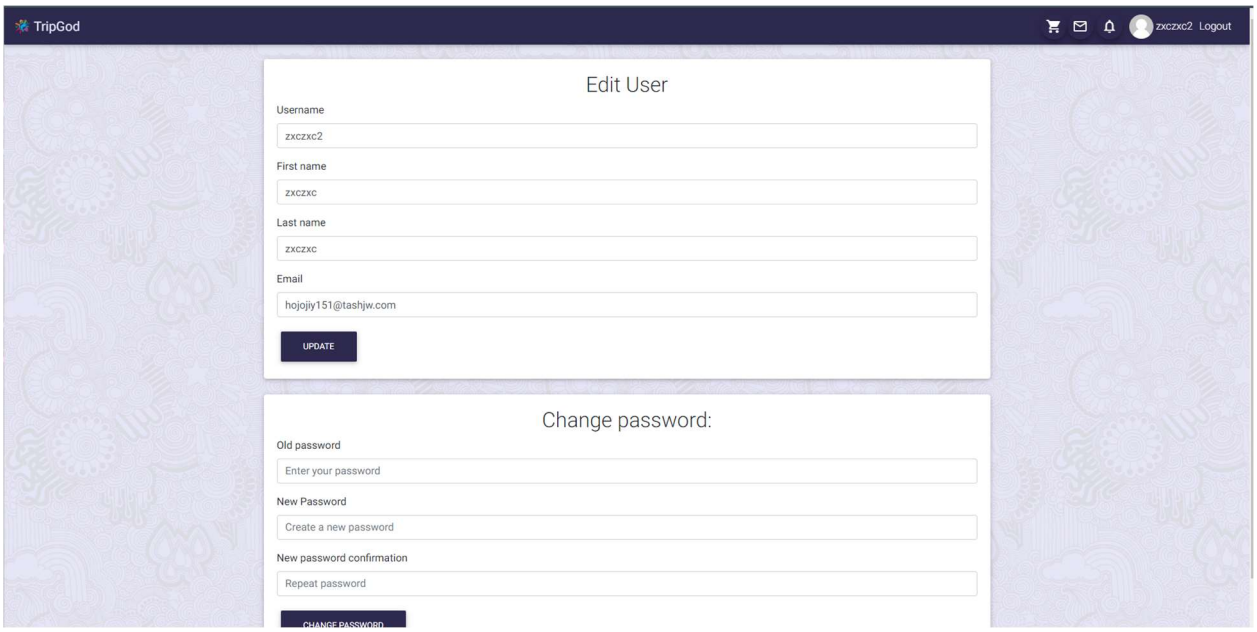


Рис 4.4. Сторінка редагування профілю

5) Далі користувачу необхідно вибрати поїздку та додати її до кошику (рис. 4.5.)

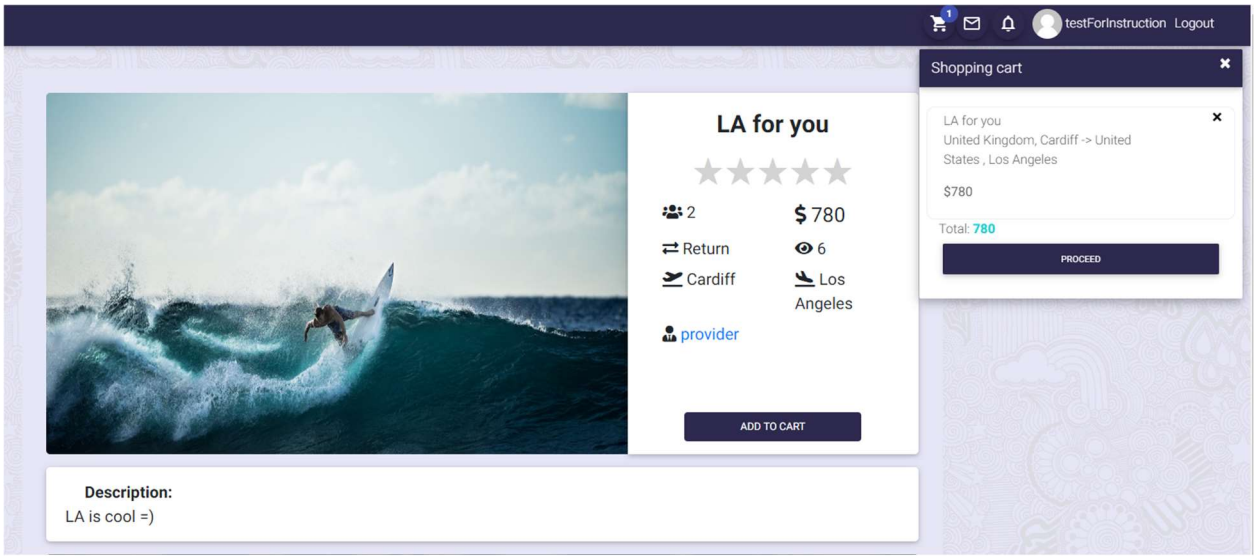


Рис 4.5. Додавання товару до кошику

6) Наступним кроком є підтвердження покупки та оплата (рис. 4.6.)

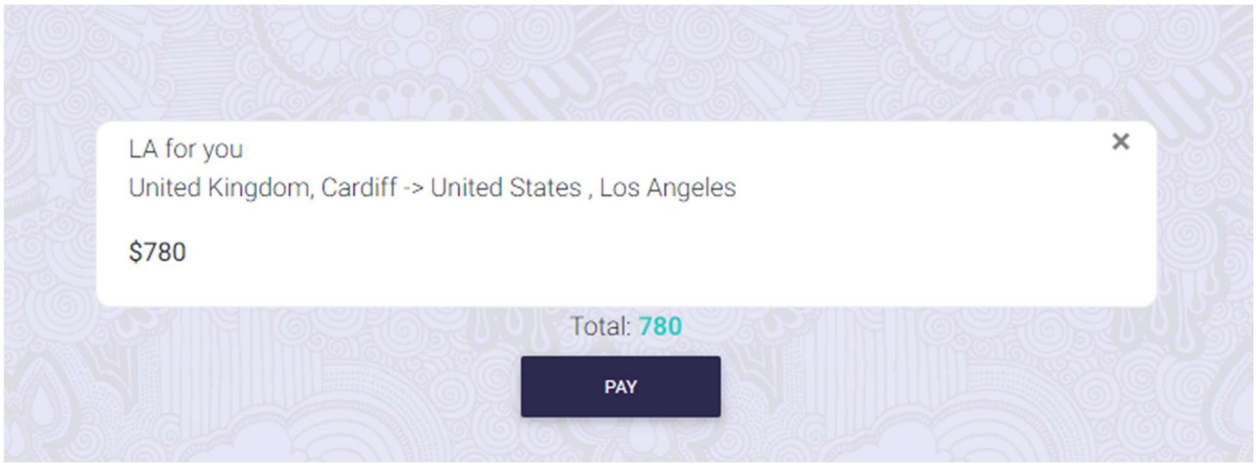


Рис 4.6. Оплата поїздки

7) Після оплати з’являється можливість залишити коментарі (рис. 4.7) для подорожі чи поскаржитись на неї (рис. 4.8.)

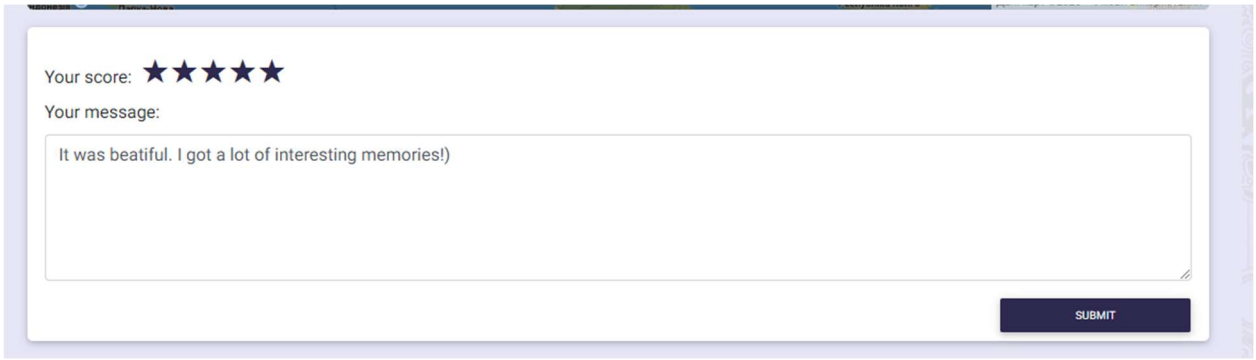


Рис 4.7. Додавання коментарів

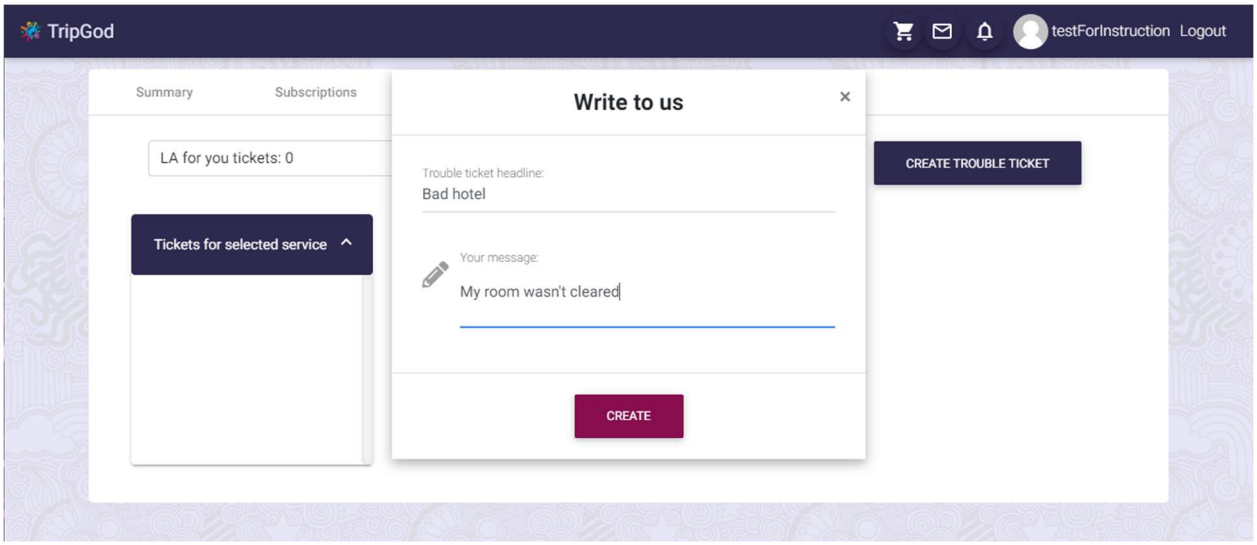


Рис 4.8. Створення скарги.

ВИСНОВОК ДО РОЗДІЛУ 4

В цьому розділі була описана «Інструкція користувача» для реалізованої клієнт-серверної системи. Детальність інструкції дозволить швидко розібратись в можливостях веб-додатку та ефективно користуватись ним.

ВИСНОВКИ

Під час виконання дипломного проекту були досягнуті поставлені цілі, а саме:

- проведений аналіз існуючих аналогів;
- на основі отриманих даних були сформульовані основні вимоги на основі яких розроблялась система;
- виходячи з отриманих критеріїв були вибрані відповідні технології які дозволили якісно виконувати поставлені задачі;
- реалізована клієнт-серверна система.

Після завершення розробки система була протестована та перевірена на працездатність.

Підсумовуючи проведену роботу можна зробити висновок, що маючи певний набір універсальних програмних модулів та компонентів, та навичок роботи з ними, можна створювати якісні продукти, котрі можуть існувати окремо, або бути інтегровані у сторонні системи.

Одержані результати дають підстави вважати, що завдання реалізовані, мета досягнута.

Література

1. Ajax: A New Approach to Web Applications [Електронний ресурс] – Режим доступу до ресурсу: http://adaptivepath.org/ideas/ajax-new-approach-web-applications
2. Why Build a Progressive Web App? [Електронний ресурс] - Режим доступу до ресурсу: https://www.youtube.com/watch?v=I10NsB4TXtw
3. PostgreSQL: альтернатива системам управління базами даних з відкритим кодом [Електронний ресурс] - Режим доступу до ресурсу: https://pdfs.semanticscholar.org/c95a/fa04bc1b8d6cd7273349948fad50995594b9.pdf
4. Erick S. R. The Cathedral & the Bazaar / Stiven Raymond Erick., 1999. – 241 с.
5. The Java™ Virtual Machine Specification — Спецификация JVM [Електронний ресурс]. – 2014. – Режим доступу до ресурсу: https://docs.oracle.com/javase/specs/.
6. Офіційний сайт PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/.
7. Spring Framework Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.spring.io/spring/docs/current/spring-framework-reference/.
8. Адам Ф. Angular для профессионалов / Фримен Адам., 2018. – 800 с. – (Для профессионалов).
9. Boris C. Programming TypeScript / Cherny Boris., 2019. – (1).
10. altrus. PWA — это просто [Електронний ресурс] / altrus. – 2018. – Режим доступу до ресурсу: https://habr.com/ru/post/418923/.
11. Server-side rendering (SSR) with Angular Universal [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/guide/universal.

12. Venkata K. Использование Angular для Single Page Applications (SPAs) [Електронний ресурс] / K. Venkata, A. Scott. – 2016. – Режим доступу до ресурсу: https://dotnet.today/ru/aspnet5-vnext/client-side/angular.html.
13. Ben A. Spring Security Reference [Електронний ресурс] / A. Ben, T. Luke, W. Rob. – 2020. – Режим доступу до ресурсу: https://docs.spring.io/spring-security/site/docs/current/reference/html5/.
14. WebSocket [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://learn.javascript.ru/websocket.
15. Dominik T. What are Web Sockets? [Електронний ресурс] / Tarnowski Dominik. – 2017. – Режим доступу до ресурсу: https://medium.com/@td0m/what-are-web-sockets-what-about-rest-apis-b9c15fd72aac.
16. supernapalm. Развертываем свой сайт на Heroku [Електронний ресурс] / supernapalm. – 2014. – Режим доступу до ресурсу: https://habr.com/ru/post/232679/.

ДОДАТОК 1

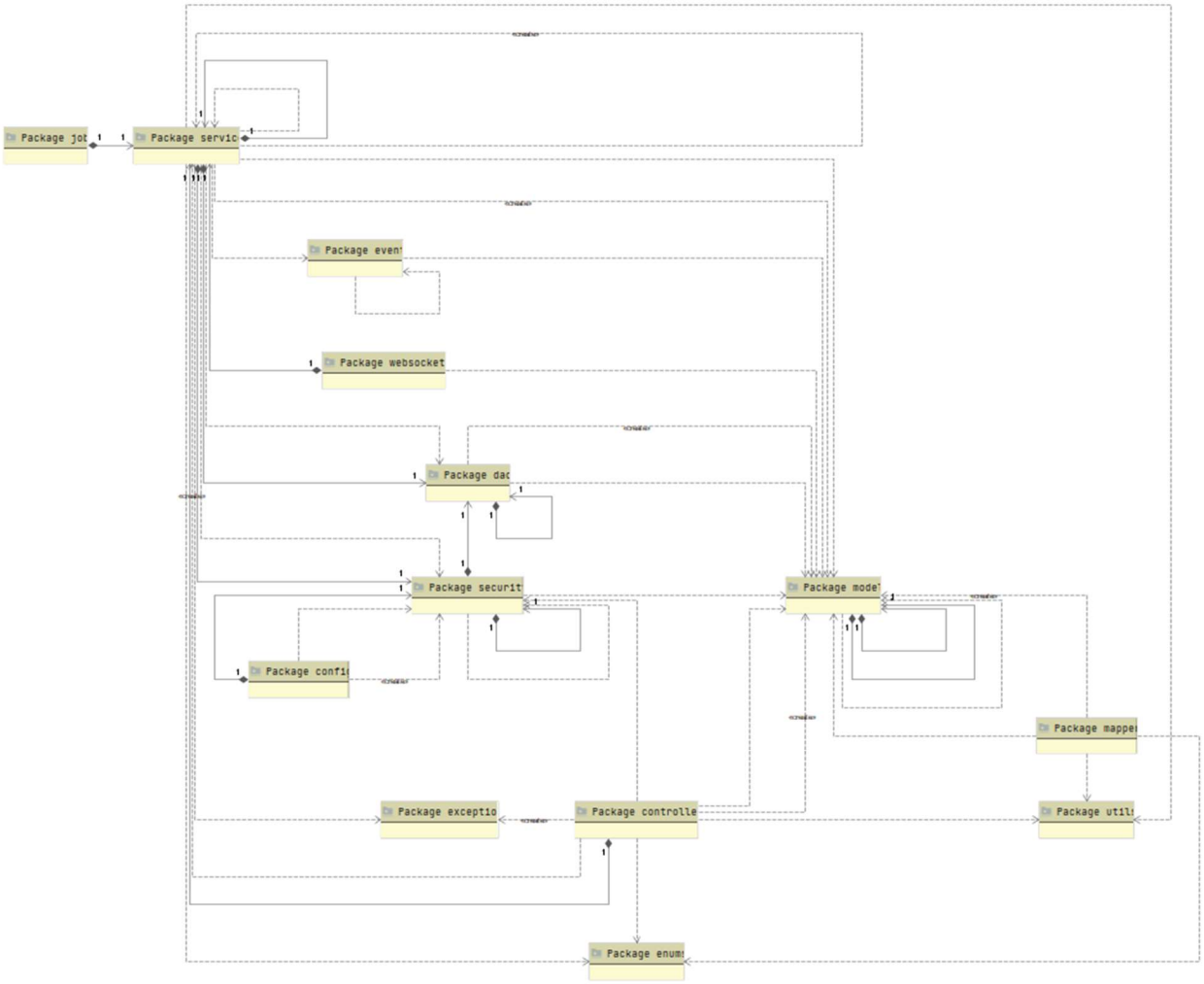
Клієнт-серверна система для продажу туристичних турів

Діаграма класів. Структурна схема.

ІАЛЦ.467200.004 Д1

Листів 1

2020



					ІАЛЦ.467200.004 Д1		
Змн.		ПІБ	Підпис	Дата			
Розробив		Мельник І.М.			Клієнт-серверна система для продажу туристичних турів. Додаток 1	Літ.	Арк. 1 / Акрушів 1
Перевірив		Алещенко О.В					
						КПІ ім. Ігоря Сікорського ФІОТ ІО-63	
Н/Контр.		Сімоненко В.П.					
Зав.каф.		Стіренко С.Г.					

ДОДАТОК 2

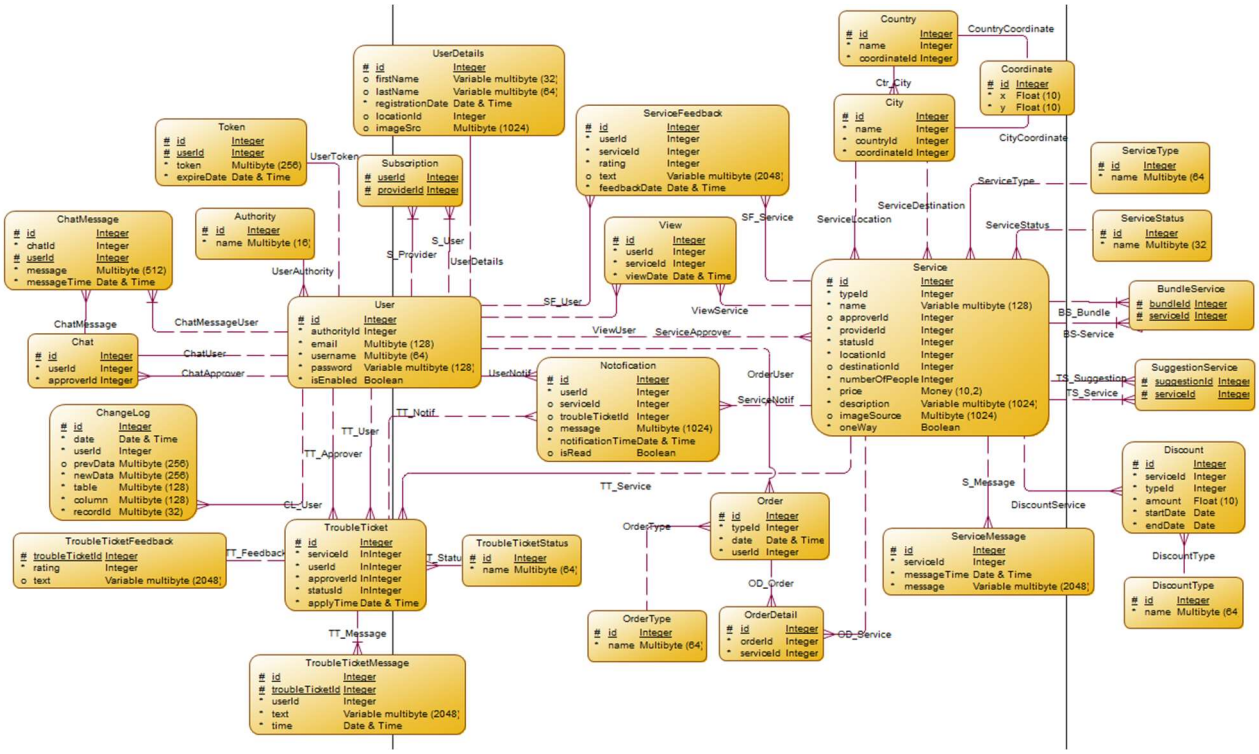
Клієнт-серверна система для продажу туристичних турів

ER-модель

ІАЛЦ.467200.005 Д2

Листів 1

2020



					ІАЛЦ.467200.005 Д2		
Змн.		ПІБ	Підпис	Дата			
Розробив		Мельник І.М.			Клієнт-серверна система для продажу туристичних турів. Додаток 2	Літ.	Арк. 1 / Акрушів 1
Перевірив		Алещенко О.В					
						КПІ ім. Ігоря Сікорського ФІОТ ІО-63	
Н/Контр.		Сімоненко В.П.					
Зав.каф.		Стіренко С.Г.					

ДОДАТОК З

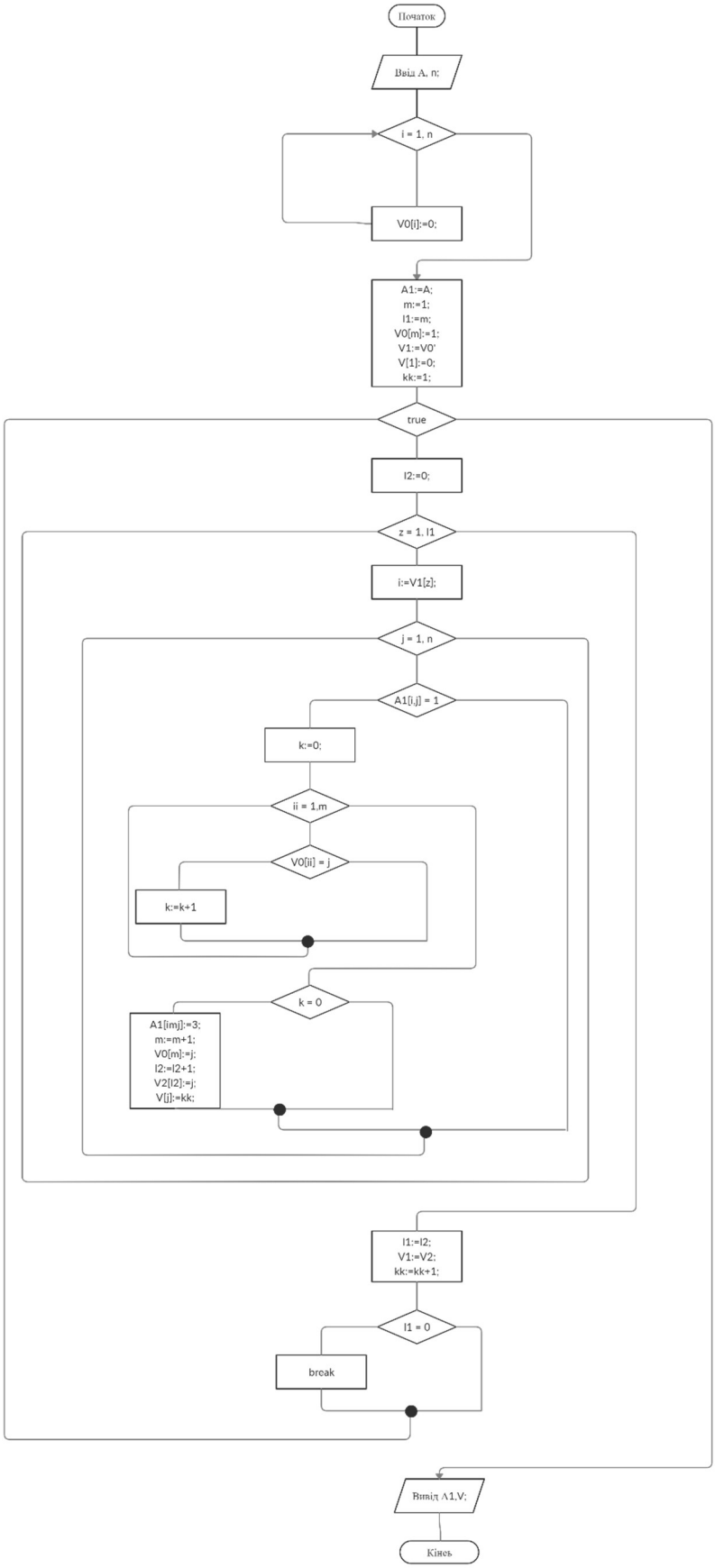
Клієнт-серверна система для продажу туристичних турів

Алгоритм обходу графа в ширину

ІАЛЦ.467200.006 Д3

Листів 1

2020



					ІАЛЦ.467200.006 ДЗ		
Змн.		*ПІБ*	*Підпис*	*Дата*			
Розробив		*Мельник І.М.*			*Клієнт-серверна система для продажу туристичних турів. Додаток 3*	*Літ.*	*Арк.* 1 / *Акрушів* 1
Перевірив		*Алещенко О.В*					
						КПІ ім. Ігоря Сікорського ФІОТ ІО-63	
Н/Контр.		*Сімоненко В.П.*					
Зав.каф.		*Стіренко С.Г.*					

ДОДАТОК 4

Клієнт-серверна система для продажу туристичних турів

Лістинг програми

ІАЛЦ.467200.007 Д4

Листів 10

2020

```
import lombok.Data;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import javax.mail.MessagingException;
import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.Objects;

@Service
@Slf4j
@Transactional
public class TripService {

   private ServiceDao serviceDao;
   private ServiceService serviceService;
   private SuggestionService suggestionService;
   private LocationService locationService;
   private DiscountService discountService;
   private LifecycleEventPublisher publisher;
   private CoordinateService coordinateService;
   private SubscriptionService subscriptionService;
   private EmailService emailService;

   @Value("${db.service.searchByCountry}")
   private String sqlSearchByCountry;

   @Autowired
   public TripService(ServiceDao serviceDao, ServiceService serviceService, SuggestionService
suggestionService,
                      LocationService locationService, DiscountService discountService, LifecycleEventPublisher
publisher, CoordinateService coordinateService, SubscriptionService subscriptionService, EmailService
emailService) {
      this.serviceDao = serviceDao;
      this.serviceService = serviceService;
      this.suggestionService = suggestionService;
      this.locationService = locationService;
      this.discountService = discountService;
      this.publisher = publisher;
      this.coordinateService = coordinateService;
      this.subscriptionService = subscriptionService;
      this.emailService = emailService;
   }

   public com.netcracker.edu.backend.model.Service getTripByIdDetailedWithSuggestions(long id) {
      log.debug("Getting trip");

      com.netcracker.edu.backend.model.Service trip = serviceDao.getTripByIdDetailed(id)
            .orElseThrow(() -> {
               log.error("Trip with id {} not found", id);
               return new AppException("Trip with id " + id + " not found");
            });

      List<com.netcracker.edu.backend.model.Service> suggestions = serviceDao.getSuggestionsByTripId(id);

      for (com.netcracker.edu.backend.model.Service suggestion : suggestions) {
```

```
            List<com.netcracker.edu.backend.model.Service> services =
                serviceDao.getServicesBySuggestionId(suggestion.getId());

            suggestion.setServices(services);
        }

        trip.setSuggestions(suggestions);

        if (trip.getLocation() != null) {
            trip.getLocation().setCoordinate(coordinateService.getLocById(trip.getId()));
            trip.getDestination().setCoordinate(coordinateService.getDestById(trip.getId()));
        }
        return trip;
    }

    public com.netcracker.edu.backend.model.Service getTripById(long id) {
        log.debug("Getting trip");

        return serviceDao.getTripById(id)
                .orElseThrow(() -> {
                    log.error("Trip not found with id: {}", id);
                    return new AppException("Trip with id " + id + " not found");
                });
    }

    public com.netcracker.edu.backend.model.Service createTrip(com.netcracker.edu.backend.model.Service trip)
{
        log.debug("Creating trip");

        return serviceDao.insert(trip);
    }

    public void updateTrip(com.netcracker.edu.backend.model.Service trip) {
        log.debug("Updating trip");

        serviceDao.update(trip);
    }

    public void deleteTripById(long id) {
        log.debug("Deleting trip");

        serviceDao.deleteById(id);
    }

    public void deleteTrip(com.netcracker.edu.backend.model.Service trip) {
        serviceDao.delete(trip);
    }

    public List<com.netcracker.edu.backend.model.Service> getAllTrips() {
        log.debug("Getting trips");

        return serviceDao.getAllTrips();
    }

    public List<com.netcracker.edu.backend.model.Service> getAllTripsOfStatus(String status) {
        log.debug("Getting trips");

        return serviceDao.getAllTripsOfStatus(status);
    }

    public List<com.netcracker.edu.backend.model.Service> getTripsByProviderId(long id) {
        log.debug("Getting trips");
```

```
        return serviceDao.getTripsByProviderId(ServiceType.TRIP.getServiceTypeName(), id);
    }

    public List<com.netcracker.edu.backend.model.Service> getTripsByProviderIdWithLocationsAndStatus(long
id) {
        log.debug("Getting trips");

        return serviceDao.getTripsByProviderIdWithLocationAndStatus(id);
    }

    public List<com.netcracker.edu.backend.model.Service>
getTripsByApproverIdOrNotAssignedWithLocationsAndStatus(long id) {
        log.debug("Getting trips");

        return serviceDao.getTripsByApproverIdOrNotAssignedWithLocationAndStatus(id);
    }

    public List<com.netcracker.edu.backend.model.Service> getTripsDependingOnAuthority(UserPrincipal user)
{
        String currentAuthority = user.getAuthorities()
                .stream()
                .map(GrantedAuthority::getAuthority)
                .findFirst().orElse("UNKNOWN");

        List<com.netcracker.edu.backend.model.Service> trips = new ArrayList<>();

        if (currentAuthority.equals(RoleName.ROLE_PROVIDER)) {
            trips = getTripsByProviderIdWithLocationsAndStatus(user.getId());
        } else if (currentAuthority.equals(RoleName.ROLE_APPROVER)) {
            trips = getTripsByApproverIdOrNotAssignedWithLocationsAndStatus(user.getId());
        }

        return trips;
    }

    public List<com.netcracker.edu.backend.model.Service> getTripsOfStatusByProviderId(String status, long id)
{
        log.debug("Getting trips");

        return serviceDao.getTripsOfStatusByProviderId(status, id);
    }

    public List<com.netcracker.edu.backend.model.Service> getTripsByApproverId(long id) {
        log.debug("Getting trips");

        return serviceDao.getTripsByApproverId(id);
    }

    public List<com.netcracker.edu.backend.model.Service> getTripsOfStatusByApproverId(String status, long
id) {
        log.debug("Getting trips");

        return serviceDao.getTripsOfStatusByApproverId(status, id);
    }

    public long countAllTrips() {
        log.debug("Counting trips");

        return serviceDao.countAllTrips().orElse(0L);
    }

    public List<CountryTrips> getTripsAmountByCountry() {
```

```
        log.debug("Getting trips");

        return serviceDao.getTripsAmountByCountry();
    }

    public List<com.netcracker.edu.backend.model.Service> getAllPublishedTripsWithDetails() {
        log.debug("Getting published trips");

        return serviceDao.getPublishedWithLocationsAndStatus(ServiceType.TRIP.getServiceTypeName());
    }

    public com.netcracker.edu.backend.model.Service
createOrUpdateTrip(com.netcracker.edu.backend.model.Service trip, long userId) {
        log.debug("Creating (or updating) trip");

        final long oldStatusId = trip.getStatusId();

        String statusToSet;

        if (ServiceStatus.DRAFT.getServiceStatusName().equals(trip.getStatus())) {
            statusToSet = ServiceStatus.DRAFT.getServiceStatusName();
        } else {
            statusToSet = ServiceStatus.OPEN.getServiceStatusName();
        }

        trip.setStatus(statusToSet);
        restoreMappings(trip); // Setting id's from string values

        if (trip.getId() == 0) {
            trip = serviceDao.insert(trip);
        } else {
            serviceDao.update(trip);
        }

        for (com.netcracker.edu.backend.model.Service suggestion : trip.getSuggestions()) {

            suggestion.setStatus(statusToSet);
            suggestion.setProviderId(trip.getProviderId());

            suggestion.setLocationId(trip.getLocationId());
            suggestionService.restoreMappings(suggestion); // Setting id's from string values

            if (suggestion.getId() == 0) {
                suggestion = serviceDao.insert(suggestion);
                suggestionService.addSuggestionLink(suggestion.getId(), trip.getId());
            } else {
                serviceDao.update(suggestion);
            }

            for (com.netcracker.edu.backend.model.Service service : suggestion.getServices()) {

                service.setStatus(statusToSet);
                service.setProviderId(suggestion.getProviderId());

                service.setLocationId(trip.getLocationId());
                serviceService.restoreMappings(service); // Setting id's from string values

                if (service.getId() == 0) {
                    service = serviceDao.insert(service);
                    suggestionService.addSuggestionLink(suggestion.getId(), service.getId());
                } else {
                    serviceDao.update(service);
                }
```

```
        }
    }
    setApproverIdOfAllServices(0L, trip); // Setting approver id to 0 null everywhere

    publisher.publishLifecycleEventNotification(userId, trip, trip.getStatusId(), oldStatusId);

    return trip;
  }


  public List<List<com.netcracker.edu.backend.model.Service>> searchedByLength(SearchTripMultipleRequest
searchTripMultipleRequest) {
    List<List<com.netcracker.edu.backend.model.Service>> result = roadSearch(searchTripMultipleRequest,
true);

    result.sort((firstService, secondService) -> {

      Double firstDistance = 0D;
      Double secondDistant = 0D;

      for (com.netcracker.edu.backend.model.Service trip : firstService
      ) {

        firstDistance += getDistance(trip);
      }
      for (com.netcracker.edu.backend.model.Service trip : secondService
      ) {

        secondDistant += getDistance(trip);
      }

      return firstDistance.compareTo(secondDistant);
    });
    return result;
  }

  public List<List<com.netcracker.edu.backend.model.Service>>
searchedByBundle(SearchTripMultipleRequest searchTripMultipleRequest) {

    roadSearch(searchTripMultipleRequest, true);
    return roadSearch(searchTripMultipleRequest, true);
  }


  private int checkRoadWithMaxDestination(List<Way> resultRoads, List<Long> destId,
List<com.netcracker.edu.backend.model.Service> allTrips) {
    int maxDestination = 0;

    for (Way resultRoad : resultRoads) {
      List<Long> currDestination = new ArrayList<>();
      for (int j = 0; j < resultRoad.getId().size(); j++) {
        //log.debug("    {}  {} - {}", resultRoads.get(k).getId().get(j),
allTrips.get(resultRoads.get(k).getId().get(j)).getLocation().getCountry().getName(),
allTrips.get(resultRoads.get(k).getId().get(j)).getDestination().getCountry().getName());
        if (destId.contains(allTrips.get(resultRoad.getId().get(j)).getDestination().getCountry().getId())) {
          if
(!(currDestination.contains(allTrips.get(resultRoad.getId().get(j)).getDestination().getCountry().getId()))) {
            currDestination.add(allTrips.get(resultRoad.getId().get(j)).getDestination().getCountry().getId());
          }
        }
        //resultRoads.get(k).getId().set(j, (int) allTrips.get(resultRoads.get(k).getId().get(j)).getId());
        resultRoad.setValue(currDestination.size());
```

```
            if (currDestination.size() > maxDestination) {
                maxDestination = currDestination.size();
            }
        }
    }

    return maxDestination;
}

 private List<List<com.netcracker.edu.backend.model.Service>>
getAllListWithoutDublicate(List<List<com.netcracker.edu.backend.model.Service>> allTrips) {
    List<List<com.netcracker.edu.backend.model.Service>> resultSet = new ArrayList<>();

    for (List<com.netcracker.edu.backend.model.Service> service : allTrips) {
        if (!(contains(resultSet, service))) {
            resultSet.add(service);
        }
    }

    return resultSet;
}


 //Breadth-first search with remembering the way
 private void getAllRoads(List<Way> resultRoads, List<Way> allRoads, int i, int length, double[][] graph,
List<com.netcracker.edu.backend.model.Service> allTrips, long locId, List<Long> destId) {
    resultRoads.addAll(allRoads);

    while (allRoads.size() - i > 1) {
        Way curr = allRoads.get(i);
        //check if there are adjacent vertices
        int lastVertex = curr.getId().get(curr.getId().size() - 1);
        int numOfVertex = 0;
        for (int j = 0; j < length; j++) {

            if (graph[lastVertex][j] != 0.0) {

                //if found cycle
                if (curr.getId().contains(j)) {
                    resultRoads.add(curr);
                } else if (checkForCicle(curr, destId, allTrips, j)) {
                    resultRoads.add(curr);
                } else if (checkForAllDestination(curr, destId, allTrips, locId)) {
                    resultRoads.add(curr);
                } else {

                    Way newWay = new Way();

                    //copy the path
                    for (int z = 0; z < curr.getId().size(); z++) {
                        newWay.getId().add(curr.getId().get(z));
                    }
                    newWay.getId().add(j);

                    //count sum
                    newWay.setValue(curr.getValue() + graph[lastVertex][j]);

                    //show that there was vertex and its not last node
                    numOfVertex++;

                    if (destId.contains(allTrips.get(j).getDestination().getCountry().getId())) {
                        resultRoads.add(newWay);
                    }
```

```
                allRoads.add(newWay);
            }
          }
        }

        //if adjacent vertices were not - added as a result
        if (numOfVertex == 0) {
          resultRoads.add(curr);
        }

        i++;
      }
    }

    //checking for 2 times in one destination
    private boolean checkForCicle(Way curr, List<Long> destId,
List<com.netcracker.edu.backend.model.Service> allTrips, int j) {
      List<Long> usedDestination = new ArrayList<>();
      for (int c = 0; c < curr.getId().size(); c++) {
        if (destId.contains(allTrips.get(curr.getId().get(c)).getDestination().getCountry().getId())) {
          usedDestination.add(allTrips.get(curr.getId().get(c)).getDestination().getCountry().getId());
        }
      }
      return usedDestination.contains(allTrips.get(j).getDestination().getCountry().getId());
    }


    private boolean checkForAllDestination(Way curr, List<Long> destId,
List<com.netcracker.edu.backend.model.Service> allTrips, long locId) {
      List<Long> usedDestination = new ArrayList<>();
      for (int c = 0; c < curr.getId().size(); c++) {
        if (destId.contains(allTrips.get(curr.getId().get(c)).getDestination().getCountry().getId())) {
          if (!usedDestination.contains(allTrips.get(curr.getId().get(c)).getDestination().getCountry().getId())) {
            usedDestination.add(allTrips.get(curr.getId().get(c)).getDestination().getCountry().getId());

          }
        }
      }

      return usedDestination.size() >= destId.size() || allTrips.get(curr.getId().get(curr.getId().size() -
1)).getDestination().getCountry().getId() == (locId);
    }

    private double[][] createAdjacencyMatrix(int length, List<com.netcracker.edu.backend.model.Service>
allTrips) {
      double[][] graph = new double[length][length];
      for (int i = 0; i < length; i++) {
        for (int j = 0; j < length; j++) {

          if (allTrips.get(i).getDestination().getCountry().getId() ==
allTrips.get(j).getLocation().getCountry().getId()) {
            graph[i][j] = allTrips.get(i).getPrice().doubleValue();
          }
        }
      }
      return graph;
    }


    public String completeSqlSearchByCountryStringWithBuffer(int numOfDestinationCountry, boolean
oneWay) {
      StringBuilder sqlSearchString = new StringBuilder(sqlSearchByCountry + " ");
```

```
        for (int i = 0; i < numOfDestinationCountry; i++) {
            sqlSearchString.insert(sqlSearchString.length() - 1, "?, ");
        }
        sqlSearchString.insert(sqlSearchString.length() - 1, "?) OR dest_country.country_name IN ( ");

        for (int i = 0; i < (numOfDestinationCountry - 1); i++) {
            sqlSearchString.insert(sqlSearchString.length() - 1, "?, ");
        }

        if (oneWay) {
            sqlSearchString.insert(sqlSearchString.length() - 1, "?)) ");
        } else {
            sqlSearchString.insert(sqlSearchString.length() - 1, "?,?)) ");
        }

        return sqlSearchString.toString();

    }

    public List<com.netcracker.edu.backend.model.Service> optimalSearch(SearchTripRequest
searchTripRequest, long userId) {
        log.debug("Searching best trip in discount by location {} and {}",
searchTripRequest.getLocationCountry(), searchTripRequest.getDestinationCountry());
        int limit = 100; //limit for trips in page and in select
        List<com.netcracker.edu.backend.model.Service> ratingTrip =
serviceDao.getSearchByRating(searchTripRequest, limit);

        log.debug("Searching best approver by user_id {}", userId);

        long providerId = 0;
        if (userId != 0) {
            providerId = serviceDao.getSearchByYourRaitingProviderId(userId).orElse((long) 0);

            if (providerId == 0) {
                providerId = serviceDao.getSearchByNumOfOrderProviderId(userId).orElse((long) 0);
            }
        }

        List<com.netcracker.edu.backend.model.Service> result = new ArrayList<>();

        if (providerId != 0) {
            for (com.netcracker.edu.backend.model.Service trip : ratingTrip) {
                if (trip.getProviderId() == providerId) {
                    result.add(trip);
                }
            }
            if (result.size() > 5) {
                return result.subList(0, 4);
            }
            return result;

        } else {
            ratingTrip.sort(
                    Comparator.comparing(
                            firstService -> firstService.getRating()
                                    .multiply(new BigDecimal(100)).subtract(firstService.getPrice())));

        }
        return ratingTrip;
    }

    public List<List<com.netcracker.edu.backend.model.Service>> searchedByPrice(SearchTripMultipleRequest
searchTripMultipleRequest) {
```

```
    List<List<com.netcracker.edu.backend.model.Service>> result = roadSearch(searchTripMultipleRequest,
true);

    result.sort((firstService, secondService) -> {
      BigDecimal firstPrice = new BigDecimal(0);
      BigDecimal secondPrice = new BigDecimal(0);

      for (com.netcracker.edu.backend.model.Service trip : firstService
      ) {
        firstPrice = firstPrice.add(trip.getPrice());
      }

      for (com.netcracker.edu.backend.model.Service trip : secondService
      ) {
        secondPrice = secondPrice.add(trip.getPrice());
      }
      return firstPrice.compareTo(secondPrice);
    });
    return result;
  }

  @Data
  static class Way {
    private List<Integer> id = new ArrayList<>();
    private double value;
  }
}
```